Creating architecture for a digital information system leveraging virtual environments

University of Oulu
Department of Information Processing Science
Master's Thesis
Matti Ollila
19.3.2019

Abstract

The topic of the thesis was the creation of a proof of concept digital information system, which utilizes virtual environments. The focus was finding a working design, which can then be expanded upon. The research was conducted using design science research, by creating the information system as the artifact. The research was conducted for Nokia Networks in Oulu, Finland; in this document referred to as "the target organization".

An information system is a collection of distributed computing components, which come together to create value for an organization. Information system architecture is generally derived from enterprise architecture, and consists of a data-, technical- and application architectures. Data architecture outlines the data that the system uses, and the policies related to its usage, manipulation and storage. Technical architecture relates to various technological areas, such as networking and protocols, as well as any environmental factors. The application architecture consists of deconstructing the applications that are used in the operations of the information system.

Virtual reality is an experience, where the concepts of presence, autonomy and interaction come together to create an immersive alternative to a regular display-based computer environment. The most typical form of virtual reality consists of a head-mounted device, controllers and movement-tracking base stations. The user's head- and body movement can be tracked, which changes their position in the virtual environment.

The proof-of-concept information system architecture used a multi-server -based solution, where one central physical server hosted multiple virtual servers. The system consisted of a website, which was the knowledge-center and where a client software could be downloaded. The client software was the authorization portal, which determined the virtual environments that were available to the user. The virtual reality application included functionalities, which enable co-operative, virtualized use of various Nokia products, in immersive environments. The system was tested in working situations, such as during exhibitions with customers.

The proof-of-concept system fulfilled many of the functional requirements set for it, allowing for co-operation in the virtual reality. Additionally, a rudimentary model for access control was available in the designed system. The shortcomings of the system were related to areas such as security and scaling, which can be further developed by introducing a cloud-hosted environment to the architecture.

Keywords
information system architecture, virtual reality, virtual environment, virtual world

Supervisor
PhD, Postdoctoral Researcher, Matti Pouke
PhD, Postdoctoral Researcher, Mikko Rajanen

Foreword

Thank you to Matti Pouke, both for his supervision and for introducing me to this thesis work opportunity in the first place. Also, thanks to Mikko Rajanen for offering his expertise in reviewing the almost finished thesis.

I would also like to thank the various people at Nokia Networks who contributed to the thesis in one way or another. Thank you to Iikka Finning and Laura Vähäsaari for hiring me, and Iikka specifically for his efforts in leading the project. Additionally, thanks to all my co-workers in the Garage innovation laboratory, who made the six months even more enjoyable. Special acknowledgement to Eetu Suni for his vast technical expertise in various areas related to the project.

Finally, I'd like to thank family and friends for the years of support. Special thanks to Jonas Olsen, because he deserves it.

Matti Ollila

Oulu, March 19, 2019

Abbreviations

AC	Access Control
AR	Augmented Reality
AV	Augmented Virtuality
CAD	Computer-aided design
CRUD	Create, Read, Update, Delete
DSR	Design Science Research
EA	Enterprise Architecture
ISA	Information Systems Architecture
MR	Mixed Reality
POC	Proof of concept
VC	Version Control
VE	Virtual Environment
VM	Virtual Machine
VR	Virtual Reality
VW	Virtual World

Contents

1. Introduction

Digitalization is an increasingly important evolution of business practices within enterprises, as previously only analogically conducted practices are transferred into bytes. (Kirchmer, Franz, Lotterer, Antonucci & Laengle, 2016.) Additionally, virtual reality has emerged as a possible next step in interactivity, and while it is mostly seen in entertainment, its possibilities in industry settings have been theorized for years (Burdea and Coiffet, 2003) and are becoming more prominent in recent years (Turner, 2017). That said, research into how such systems is built is sparse and typically very conceptual, without instantiated implementations. It's therefore a novel focus on an otherwise mature area of information systems. The problem of building such a system is studied through the following research question:

RQ: "What kind of architecture supports a digital information system leveraging virtual environments?".

Further, this research question will be supplemented with the following sub-questions (SQ):

SQ1: "What kind of data architecture supports a digital information system leveraging virtual environments?"

SQ2: "What kind of application architecture supports a digital information system leveraging virtual environments?"

SQ3: "What kind of technological architecture supports a business information system leveraging virtual environments?"

These research questions are answered using the design science research method by producing a proof-of-concept (POC) architecture for a digital information system. The contribution of this thesis is a novel implementation of established concepts of enterprise- and information system architecture frameworks, by presenting the applied design methodology, as well as the architecture specification and implementation for a POC digital information system. The implementation combines an information system architecture with a collaborative virtual reality environment. The research's main limitations are organizational restrictions, as the works is conducted in a real company setting. Therefore, company policies must be adhered to and resources are limited.

Information System Architecture (ISA) is one part of many architectures that are relevant to an organization and is a subset of enterprise architectures (EA). The point of ISA is to create and describe the order and structure of how an IT system of networked components is comprised, used and how it benefits the organization. There are several frameworks for creating enterprise architecture, which can be adapted to fit the ISA viewpoint. (Vasconcelos, Sousa & Tribolet, 2007.) Some of the better-known ones include Zachman's framework, The Open Group Architecture Framework, Federal Enterprise Architecture and The Gartner Practice (Tupper, 2011).

While the frameworks differ in certain ways, they have areas that are typical of ISAs. Data architecture describes the way data, that the organization uses in the specific information system (IS), is handled, detailing data policies and lifecycle events, as well

as structures. (Tupper, 2011.) ISA also includes a technical architecture, which is typically divided into application and technological architectures. Application architecture models the details of the applications that the IS uses, while technological architecture describes technological solutions, such as hardware, networks and computational distribution in the environment. (Vasconcelos, Sousa & Tribolet, 2007.)

The thesis is structured as follows. Chapter two introduces the prior research in the areas of information systems, enterprise architecture frameworks and concepts and the concept of virtual reality. Chapter three introduces the research method and outlines it in the context of this thesis. Chapter four outlines the empirical research and the development of the POC IS. Chapter five presents the findings, which are then discussed in chapter six. Finally, chapter seven summarizes and concludes the thesis.

2. Prior Research

In this chapter, prior research related to ISA and the relevant frameworks are established. ISAs can be defined as a subsection of EAs, having many of the same contextual areas.

2.1 Information System Architecture as a Concept

Wieringa, Blanken, Fokkinga & Grefen (2003) define architecture of a system as "the structure, or a set of structures, of a system, consisting of elements and relations between these, such that the relations between the elements create an overall coherent system with an added value for its environment." Their definition includes many kinds of systems, such as information and workflow management systems, as well as businesses and other organizations. According to Westmark (2004), IS usually manifest as "distributed network systems that consist of components of varying quality that have been integrated to provide services for the end-user". In the information technology sense, they can be thought of as various technological components that are connected in some way and share and communicate data between each other. Hevner, March & Ram (2004) also add that information systems have an inherent quality of improving business processes, bringing a financial, practical angle to the otherwise technical concept.

As Vasconcelos, Sousa & Tribolet (2007) write, ISA is one part of many architectures that are relevant to an organization. On a higher level, there is EA and on a lower, more specific level, software architecture. Software architecture is mainly concerned with the specific ways software is built and how it operates, being interested in areas such as classes and objects. EA has several definitions. Institutions, such as The Institute for Enterprise Architecture Developments sees it as "a complete expression of the enterprise; a master plan which acts as a collaboration force between aspects of business planning such as goals, visions, strategies and governance principles; aspects of business operations such as business terms, organization structures, processes and data; aspects of automation such as information systems and data bases; and the enabling technological infrastructure of the business such as computers, operating systems and networks". The Enterprise Architecture Center of Excellence states that "Enterprise Architecture explicitly describes an organization through a set of independent, non-redundant artifacts, defining how these artifacts interrelate with each other and develops a set of prioritized, aligned initiatives and road maps to understand the organization, communicate this understanding to stakeholders, and move the organization forward to its desired state. Scholars, such as Ross, Weill & Robertson (2006) define it as "the organizing logic for business processes and information technology (IT) infrastructure reflecting the integration and standardization requirements of the company's model". Taleb and Cherkaoui (2012) summarize these definitions by stating that EA has several main architectural components, such as processes, systems, technologies, components, relationships and requirements, and the means to represent them. Vasconcelos, Sousa & Tribolet (2007) describe ISA as a subset of EA, the artifacts concerning business architecture and processes being the usually missing portion. According to their definition, ISA "addresses the representation of the IS components structure, its relationships, principles and directives, with the main propose of supporting business".

2.2 Information System Architecture Frameworks

While ISA frameworks don't exist per se, different organizations and entities have created their own frameworks and guidelines for enterprise architecture creation. Tupper (2011) introduces four prominent frameworks: Zachman's framework, The Open Group Architecture Framework (TOGAF), The Federal Enterprise Architecture (FEA) and The Gartner Practice. While each framework is presented as an enterprise architecture, they can be modified to fit the ISA viewpoint, by simply ignoring the business architecture portion, as Vasconcelos, Sousa & Tribolet (2007) advise.

Zachman's (1987) framework is typically considered as the inception of ISA. Although the framework was later dubbed an EA, he viewed it as an ISA and it can still be thought of as such, with certain modifications. Zachman's challenge was related to increasingly distributed systems and his solution takes a holistic approach, where every important aspect is inspected from every important perspective. The framework was later extended by Sowa and Zachman (1992) and the extended version is what is currently referred to as "Zachman's framework". As Tupper (2011) explains, Zachman's framework divides the architecture into not only separate thematic entities, but the different stakeholders of the building process are also considered (Table 1).

Table 1. Zachman's extended enterprise architecture framework (Sowa and Zachman, 1992).

	DATA	FUNCTION	NETWORK	PEOPLE	TIME	MOTIVATION	
SCOPE PLANNER	List of things important to the business	List of processes the business performs	List of locations in which the business operates	List of organizations/agents important to the business	List of events significant to the business	List of business goals/strategy	**SCOPE** PLANNER
ENTERPRISE MODEL OWNER	E.G. "Ent/rel diagram"	E.G. "Process flow diagram"	E.G. "Logistics networks"	E.G. "Organization chart"	E.G. "Master schedule"	E.G. "Business plan"	**ENTERPRISE MODEL** OWNER
SYSTEM MODEL DESIGNER	E.G. "Data model"	E.G. "Data flow diagram"	E.G. "Distributed system architecture"	E.G. Human interface architecture	E.G. "Processing structure"	E.G. "Knowledge architecture"	**SYSTEM MODEL** DESIGNER
TECHNOLOGY MODEL BUILDER	E.G. "Data design"	E.G. "Structure chart"	E.G. "System architecture"	E.G. "Human/technology interface"	E.G. "Control structure"	E.G. "Knowledge design"	**TECHNOLOGY MODEL** BUILDER
COMPONENTS SUB-CONTRACTOR	E.G. "Data definition description"	E.G. "Program"	E.G. "Network architecture"	E.G. "Security architecture"	E.G. "Timing definition"	E.G. "Knowledge definition"	**COMPONENTS** SUB-CONTRACTOR
FUNCTIONING SYSTEM	E.G. Data	E.G. Function	E.G. Network	E.G. Organization	E.G. Schedule	E.G. Strategy	**FUNCTIONING SYSTEM**

The framework is structured as a table. The first five rows of the table represent the different actors inside an enterprise, which highlight the idea that different stakeholders require a different representation of the system. Tupper (2011) comments how Zachman's idea behind the framework is to present architecture in differing levels of abstraction and detail, as not all stakeholders are interested in the same aspects. For example, the upper management may find the high-level decision more pressing, whereas the builder of the system requires finer details of technical solutions. (Tupper, 2011.) Sowa and Zachman (1992) move on to the columns, writing how they answer to the questions of what, how, where, who, when and why; essentially, the building blocks of the system depend on these questions. The framework considers areas such as data, function, network, people, time and motivation, in relation to the organization and the developed system.

Sowa and Zachman (1992) add several rules to the use of the framework. The columns of the table are unordered, to avoid bias and the mental fallacy that certain cells being in certain locations would add unintended importance to them. Each column has a simple model, represented by the questions and areas of focus mentioned earlier, abstracting some real-world equivalent found in the system. While the columns are connected, each has a unique model and represents a unique perspective. This comes from the stakeholder need for specific level of abstraction. Related to the uniqueness of the columns, the cells are also unique, meaning that each entity of the framework can only exist in a single cell. As Tupper (2011) notes, if the cells cannot be made to consist of only unique representations, the problem is most likely with the designed artifact. Sowa and Zachman (1992) continue to the rows, ruling that the composite of all cell models comes together to create the model for the entire row. In other words, row is a sum of its parts, these parts being the cells. If new cells are added, they must adhere to the logic of the overall perspective of the row, while still being independent of one another. Sowa & Zachman conclude by stating that the logic of the framework is recursive, meaning that it can be used to describe any kind of system that has an owner, a designer and a builder, which highlights the holistic approach.

As Tupper (2011) writes, TOGAF divides EA into four sections, these being business, application, data and technical architectures (Figure 1). It therefore takes a directly comparable approach to Vasconcelos, Sousa & Tribolet's (2007) ISA model.

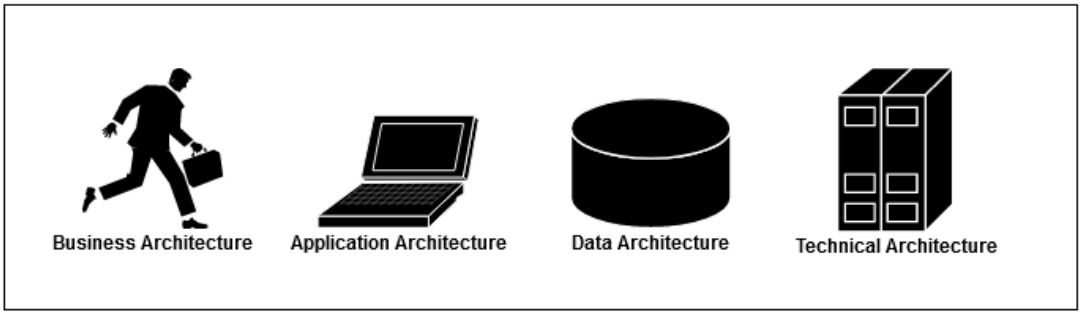

Figure 1. TOGAF framework (Tupper, 2011).

Business architecture encapsulates the business processes that the organization uses to meet its business goals. Application architecture describes how the different applications are designed and their cross-interactions. Data architecture describes the data the organization uses and how it is structured. Finally, technical architecture describes both the hardware and the software that support the applications and their interactions. (Tupper, 2011.) Taleb and Cherkaoui (2012) consider TOGAF a mature practice, with many organizations having adopted it. For example, Yuriana and Rahardjo (2016) describe an agile EA for a mining company using TOGAF.

However, the more visible aspect of TOGAF is its architecture design method (ADM), which is a specified process for how to create architecture (Figure 2).

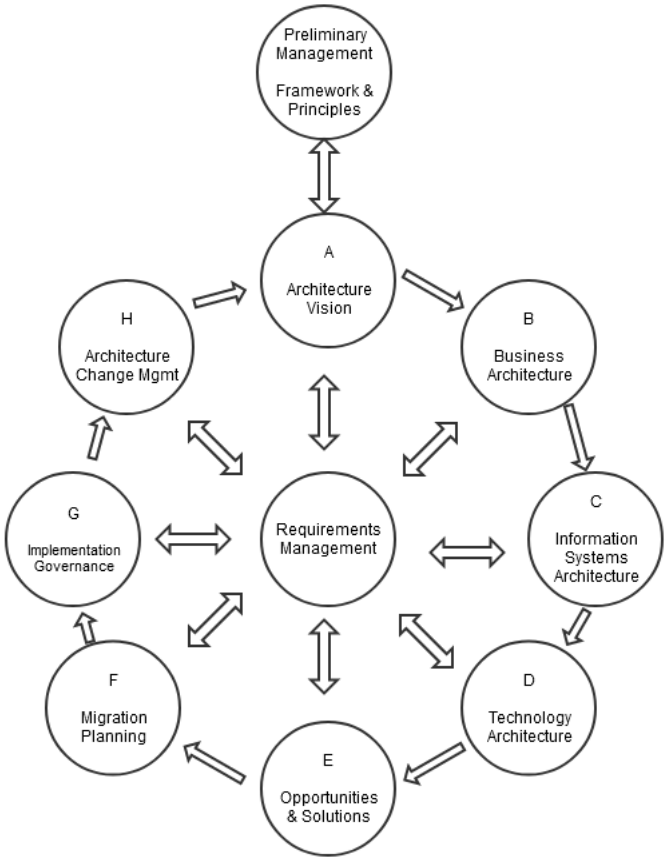

Figure 2. TOGAF architecture design method (Tupper, 2011).

Starting with initial specifications of principles and the architectural vision, it goes through the four architectural sections. Afterwards, the focus shifts to choosing and evaluating the available projects, as well as planning and overlooking the architectural migration. Finally, any changes that are required after these actions are documented and implemented before the process starts anew. At the heart of the method is the requirements management, which is the continuous process of making sure that any activities follow set requirements. Despite being a very holistic in its description, TOGAF is intended to be used in a context-specific manner. In other words, the model and its steps can be rearranged and reordered, customized, combined or skipped entirely, all depending on what the organization building the architecture requires. That said, TOGAF is also characterized by its absence of guidelines for what kind of artifacts the process should deliver. (Tupper, 2011.)

TOGAF can be used in conjunction with Zachman's framework. Zachman offers a way to categorize architectural artifacts, while TOGAF offers a method for creating them. Taleb and Cherkaoui (2012) introduce a pattern-based approach, where different architectural slices are divided into blocks and visualized using different patterns, to take advantage of pattern-based design and reuse. TOGAF also defines different layers of architectures in a single, interconnected continuum (Figure 3)., from generic architectures to the specific ones. (Tupper, 2011.)

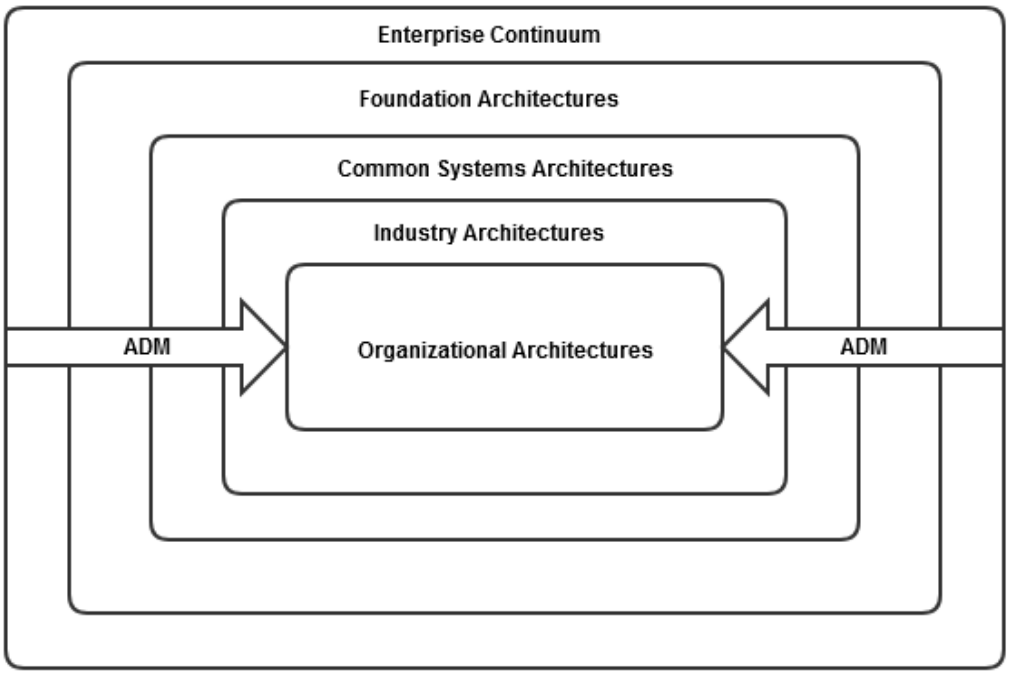

Figure 3. TOGAF enterprise continuum (Tupper, 2011).

The continuum develops from the outer layers inward. The most generic architectures are called foundational. Common systems architectures are ones that can be found in many, but not necessarily all EAs. Industry architectures are ones that are specific to certain industry branches, such as pharmaceutical companies. The inmost layer is the organizational architectures, which are enterprise-specific. (Tupper, 2011.) For example, this can relate to how an organization handles its workforce, applications and so on.

The Federal Enterprise Architecture (FEA) was designed by the US federal government, to unify its different agencies under a common EA. FEA divides an organization into segments, one segment being a major organizational entity, such as human relations. The goal is to develop a common business language between the different entities. FEA consists of five reference models, as shown in Figure 4. (Tupper, 2011.) FEA's goal is to organize and promote the sharing of critical information within the Federal Government (Urbaczewski and Mrdalj, 2006).

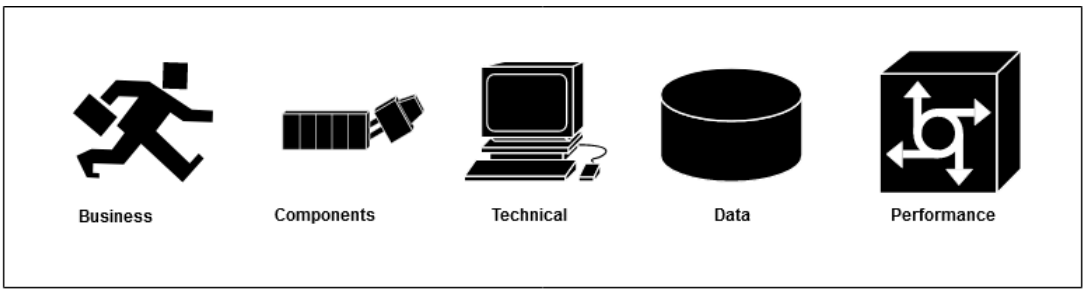

Figure 4. FEA architecture reference models (Tupper, 2011).

The reference models include models for business, components, technical, data and performance aspects of the architecture, which are analogous to how the Zachman's framework and TOGAF are constructed. These models are implemented using a four-step process (Figure 5). (Tupper, 2011.)

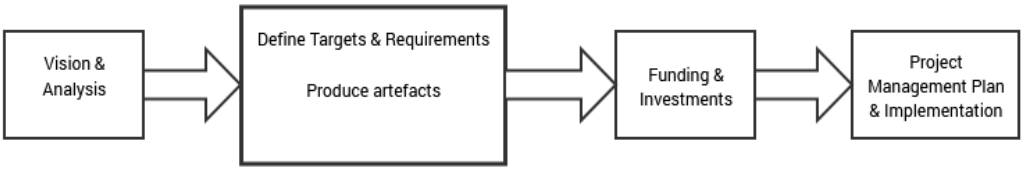

Figure 5. FEA architecture process (Tupper, 2011).

FEA process is a compact version of TOGAF and relates to a specific segment of the enterprise. The process starts with analysis, which is the preliminary vision. The second step is the bulk of the architecting work, from defining targets and requirements, to specific architectural artifacts. The third step concerns the funding of programs and the necessary investments for the architectural implementation. The final step is to plan project management and execute architectural implementation projects. (Tupper, 2011.)

The Gartner Practice, as its name suggests, is a significantly different approach to creating an enterprise architecture. Instead of being interested in a specific method, the enterprise simply hires a consultant from the Gartner organization. The idea is that a specific framework is not the key, but the reputation and quality of the work. The Gartner consultant will undoubtedly use similar ways of getting the architecture done as what Zachman's framework, TOGAF and FEA describe, but they're not highlighted in a similar fashion. Still, Gartner has some fundamental practices, that it deems timeless. Constant iteration is one of their major focuses; architecture should be a continuous process, instead of a myriad of byproduct artifacts that are soon forgotten. Unity between different organizational entities is another aspect Gartner aims toward, as they deem the common ground between business owners, information specialist and technical implementers a key toward achieving a successful architecture. This also relates to Gartner's emphasis on the target and the ways getting there, rather than the current situation. The focus of the consultant is typically on the bigger picture, offering fine grained solutions only when necessary, keeping the business needs as the driving force. (Tupper, 2011.) In many scenarios this approach to architecture isn't either feasible or appropriate, but it highlights how outsourcing the whole architectural process, including the framework, can be a viable alternative. This practice can also be taken as including some good practices in architecting, such as iteration and unity.

Tupper (2011) highlights how the differences and similarities in the frameworks can be a detriment and a benefit, as they can create confusion as to how one should proceed, while simultaneously offering opportunities for picking and choosing an exact method. Urbaczewski and Mrdalj (2006) conducted a comparison review of the different methods and add that the different levels of abstraction between the methods can propose a dilemma on the side of the architect, as the uncertainty of how to proceed exactly within a framework can be brought into question. They conclude that while Zachman's framework appears to be the most comprehensive, most of the frameworks don't explicitly represent a large portion of possible viewpoints. Additionally, the inherent difference between the frameworks makes it difficult to compare them.

2.3 Data Architecture

Data architecture is a universally implemented area of ISAs, and similarly, different sources offer similar definitions for what data architecture is. According to Lewis (2001), data architecture is the definition of how "data is stored, managed and used in a system". As he further expands, data architecture should explain data referencing and manipulation by the system's components and processes, as well as the relationship

between data and legacy systems. Namely, how these systems access the data and how their interfaces to data are implemented. Finally, he mentions the implementation of common data operations as a part of data architecture. Wu (2009) agrees, stating that data architecture should describe both the data structures and the methods of processing, storing and utilizing the data. Tupper (2011) highlights the importance of understanding how data in and of itself means different things to different entities in the organization. To a business owner, data encompasses actors that have a part in the business process, such as customers and customer demographics, and products and inventories. To a database designer, however, data is values in columns and rows, that have mathematical dependencies and connections. While these two are vastly different definitions, some sort of connection should remain between the two. For example, the contents of a database should be in line with the higher-level requirements that a business owner has for a business process. Additionally, it's important to realize that neither perspective is superior to the other and that they complement each other; knowing the context of the language to use is vital for proper data architecture design.

The importance of data architecture is also stressed, both in terms of operational and financial gains. Lewis (2001) argues, that without a data architecture implementation, the system can be exposed to several problems. Without a uniform system across the organization, common data operations may end up with differing implementations, which not only hinders present-day control of the data, but also trying to apply predictive analysis of data flow. Additionally, different standards can lead to different requirements, leading to possible design problems. Tupper (2011) challenges the notion that an architecture would cause stiffness in the overall process, instead arguing that it leads to stability. If the architecture is constructed in such a way that it can evolve, it will remain flexible and modifiable in the future. To achieve this, there must be a clear definition of how the business environment can evolve and how the architecture can help in solving the user's problems.

Data architecture is a multi-faceted and extensive topic and for that reason, it can be reviewed from several viewpoints. To go with the descriptions of data architecture as a concept, the following chapters will look at the issue in terms of data management and data lifecycle.

2.3.1 Data Policies

As Simon (2010) writes, most organizations suffer from chaotic data operations, manifesting in many of the issues outlined by Tupper (2011). For example, an organization may have a history with many different software tools and business applications, spread across many entities. These entities, with no formal framework in place, will develop their data operations in isolation, resulting in different rules, designs and implementations, which often lack the professional and tested quality and auditing that an architecture would alleviate. As a result, data becomes difficult to work with, as it exists in a fragmented and inconsistent form. Tupper (2011) outlines eight principles for data policy, to ensure proper data management.

1. Integrity in assembly and maintenance
2. Renewable and reusable assets in data development and housing
3. Highest possible data quality & integrity
4. Storage optimization for utilization and safekeeping
5. Defined ownership and custody
6. Industry-wide standards for data handling and processing
7. Data captured, validated and scrubbed at the earliest point in the process
8. Data sharing is encouraged and fostered

Data integrity is the principle of assuring the correctness of the data and it should be guaranteed in data creation and when it's being maintained. Integrity, as well as lack of it, affect both the intra-organizational activities, but the customer-facing side of the business process, as they too rely on the data being accurate. This policy goes together with aiming to achieve the highest possible data quality and integrity, which supports an organization's needs in a dynamic business environment where there is often a need for swift decisions. One step to ensure data integrity is to clearly define the ownership of data and the custodial responsibilities. The ownership management makes it so that specific entities only access the data that is relevant to them, instead of having full access to everything. This makes it easier to monitor and ensure intended usage. Custodial duties are like that of a system administrator, someone with authority and responsibility over data access and integrity. (Tupper, 2011.) According to Wu (2009), the key to high data quality is minimizing data duplication, and to achieve this, data sharing, and reuse should be promoted, and the data architecture should be tailored to accommodate efficient data usage.

In-house methods and tools for data creation and handling should be renewable and reusable. This makes it easier to expand the project, as well as control expenditures. (Tupper, 2011.) Harsh (2008) notes how data reuse will eventually transcend into knowledge reuse and creates patterns that cross between technologies and people. He argues that knowledge reuse is a key factor in decreasing costs and efforts required in business operations and reusable knowledge implementation in new projects offers avenues for further exponential exploits. Data storage and sharing tie into data reuse and should be done in a manner that supports optimal usage and safety. This requires knowledge and research of current viable technical solutions, as well as recognizing the specific needs of the organization and the process itself. Data sharing should be actively encouraged and fostered. For the organization to have consistency and functional co-operation, it is important for different entities to be aware of the advantages of sharing the data as necessary. When multiple departments are dependent on the same data, having pre-existing and agreed upon ways of co-operation guarantees the whole process runs as expected. (Tupper, 2011.)

An important consideration in data policies, when it comes planning, is considering what has already been done in the past, as it influences all the layers of data architecting. Initially, the old needs must be acknowledged simply to ensure they are being accounted for in the future design and the important elements are carried over. In the data modeling, history can be used to derive business requirements already in place and it has an effect in practical decisions, such as choosing a database management system. History can also offer insight on data access management, as it can provide characteristics about its historical use and distribution. Data's physical location and indexing are also affected by this information. (Tupper, 2011.)

Aside from data's phases throughout its lifetime, the amount of data is also an important point of consideration. While it doesn't typically have a straightforward answer, as predicting it can be difficult, there should at least be a rough estimate. Requirements engineering and thinking of the user needs are a useful reference point, as the methods and needs of a given context are organization-dependent. (Tupper, 2011.) An area that can easily get overlooked in data architectures is metadata, which is data about data. In other words, metadata describes specific information about a given data entity, depending on what the organization requires. As different data repositories come together, and separate organizational actors start co-operating, metadata can help with certain issues, such as finding common ground in data definitions. While metadata is context-sensitive, there are some universally applicable attributes that can be

documented as metadata, such as data's value source, what changes have been performed on the given data unit and what business rules apply to a data unit. Any relationships to other data or applications can also be stored using metadata. (Simon, 2010.)

Failure to follow these policies leads to stumped organizational growth, as well the devolution of strategies and operations that the company uses. Instead of keeping or developing an advantage, their actions would shrivel and become non-competitive, unable to withstand the pressure of their competitors. On the other hand, EA acts as a foundation for all the development activities; it is the level where every moving part comes together and has a slew of benefits. Because it is so far-reaching, it offers global understanding of data operations and acts as a framework for communication. It also prevents the chaos when introducing new actors to work with the data, as the structure that architecture provides can be universally applied. Distribution leads to new actions and designs, a process that can be made more flexible by designing encapsulated data solutions, where the technology is not directly tied to the needed data itself. (Tupper, 2011.)

2.3.2 Data Lifecycle

While Ball (2012) focuses on research data lifecycle management with his review of models, the principles in use are such that they can be implemented regarding any kind of data. As he writes, the data lifecycle models help with providing a structure for how the acquired data is handled. As he adds, it is much easier to curate and preserve data, when the actions that take place are predetermined. In his review, Ball goes through several data lifecycle management models aimed at research data management. While they all differ in some ways, they also showcase the similarities in the various approaches. Figure 6 is a combination of these various models.

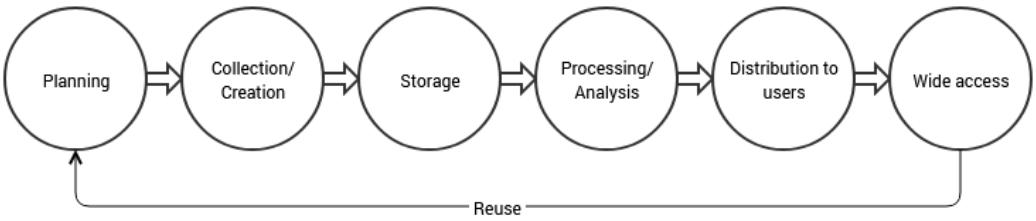

Figure 6. An applied model of data lifecycle models based on Ball (2012).

Data lifecycle management typically begins with planning, which includes defining the stakeholders, how the data will be preserved and who has access to it. Data is then collected or created, to which there are many ways in scientific practices. The raw data is allocated to a storage space and subsequently processed, so that it can be used in the intended purpose. When the data has been transformed to its intended form, it can be distributed to users and further made accessible to a wider audience. Finally, data can be re-used, for example in a new research. (Ball, 2012.)

While Ball's article is purely looking at research data, the principles can be applied to a business environment, where product data is needed to create a finished product. When data has many entities taking part in its creation and usage, it is vital to have a planned model on how the data is managed throughout its lifecycle. Tupper's (2011) consideration of data in a more generic sense also includes similar tasks during the data lifecycle, such as capture, validation, scrubbing and utilization. The tools and methods

used should be in in wide use in the industry and be well-known and accepted. This makes it both a secure solution, but also an expandable one, when new entities are integrated into the process. Additionally, data should be captured and processed at the earliest possible point in time of the process, to alleviate data quality issues that may rise in the later stages, when new functions are introduced that also use the data. The research data lifecycle management models can therefore be adapted and repurposed to better fit a business product data. Taking a strictly product-centric viewpoint, Stark (2016) establishes a five-phase lifecycle for a product (Figure 7).

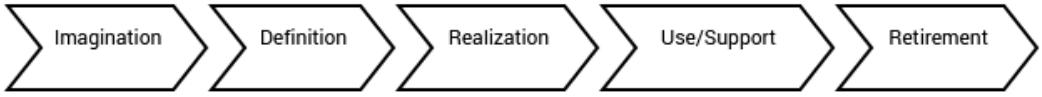

Figure 7. Five-phase product lifecycle (Stark, 2005).

The imagination phase takes place when the product is nothing more than an idea, without any concrete form. Definition phase sees these ideas transformed into an explicit form, such as a design description. Realization phase covers the process of creation, and by the end of the phase, the product exists in its final form, which can be used for its intended purpose. During the use and support phase, the product is used by the customer, while being maintained by the manufacturer. The product's lifecycle ends when it's deemed no longer useful, by the company and the customer. The company will then retire it from manufacturing and the customer will end its use. (Stark, 2016.)

Ameri and Dutta (2005) offer several ways to implement data lifecycle management in a more generic product data setting, a concept known as product lifecycle management (PLM). Streamlining is a major contributor, as both product information and creation process throughout its lifecycle can be streamlined. The idea is to have the right place, the right tool and the right time for everything. When products become more extensive, they require increased knowledge and expertise, which can be alleviated through collaboration. When this is done, regardless of location and expertise, issues such as long cycles, higher costs and low quality can be avoided. Computational frameworks can also be used, when it comes to capturing, representing, retrieving and reusing product knowledge.

PLM is a "business strategy for creating a product-centric environment", and it can bring together various product stakeholders over the entire lifecycle of the product. PLM can be defined as a set of tools and technologies that provide a shared platform for collaboration among stakeholders and streamlines the flow of information along all stages of product lifecycle. While the tools are important, the focus is on strategy and knowledge management. The technology components of PLM are the enablers of creating, transforming and sharing of knowledge and as such, they can be very rudimentary, such as a notebook, a phone or a video conferencing system. The tools are valued based on their knowledge management capabilities, instead of complexity or novelty. (Ameri and Dutta, 2005.)

Organizational activities don't exist in a vacuum and finding ways to marry mature activities with either each other or with less refined ones, is a challenge standing in the way of further growth and the so-called operational excellence. Some individual tasks may already be on a sufficient level to gain competitive advantage, but the issues rise from this necessary joint operability. Interoperability and usability therefore become quality factors for excellence. One way to reach operational excellence is to reduce

waste in value chain activities and in the linkages among them (Figure 8). (Ameri and Duta, 2005.)

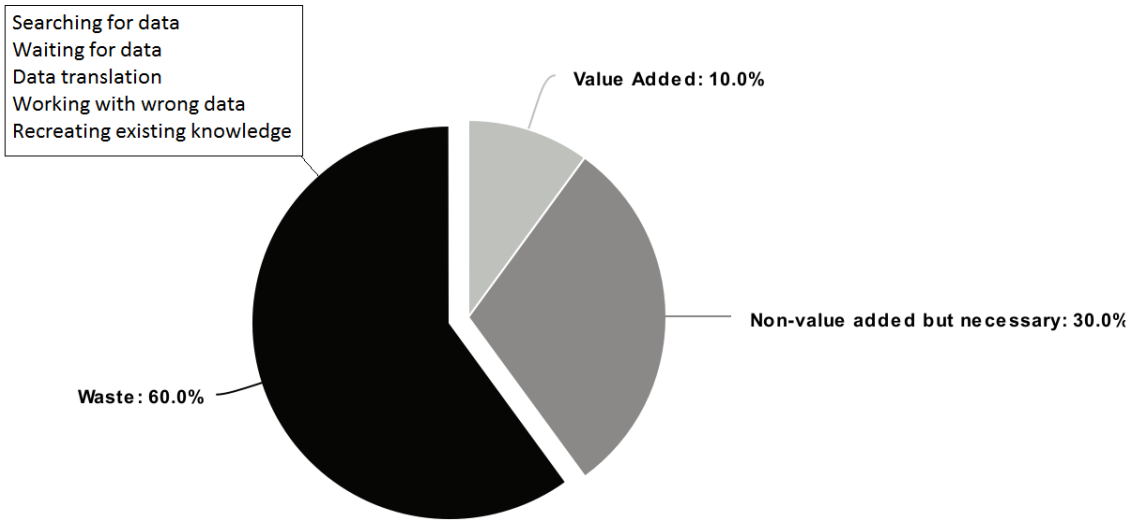

Figure 8. Usage of time in value chain activities (Ameri and Dutta, 2005).

60% of total operational time in most businesses is wasted time, which can be usually attributed to the absence of an efficient knowledge management system. Searching and waiting for data, data translation, wrong data, reinventing existing data are common.

Implementing a PLM system has the possibility for many improvements in the organization's workflow. It enables information sharing between every necessary entity, throughout the product's lifecycle. Manufacturing and engineering benefit from shortened cycle of new releases, approved improvements and implementations. Sales personnel can more effectively co-operate with suppliers. Executives can get a high-level view of all-important information related to the product. However, a PLM system can be difficult to implement on a company-scale, due to different departments using the data in different ways. This usually requires a neutral party to oversee the system implementation, to avoid biases rising from historic ways of doing work. That said, selecting individual tools, such as computer assisted design (CAD) software, can work on a hands-off basis, if the different tools can interact with each other. (Sudarsan, Fenves, Sriram & Wang, 2005.) Stark (2016) also mentions the possible challenges when managing a product throughout its lifecycle, as the responsibility can vary greatly over time and with different goals, methods and tools, a common approach can be difficult to maintain.

2.4 Technical architecture

Technical architecture of an ISA is conceptually both straightforward and fragmented, as there are several views about what should be included and at what level of abstraction and detail. Figure 9 visualizes the typical separation into application and technological architectures.

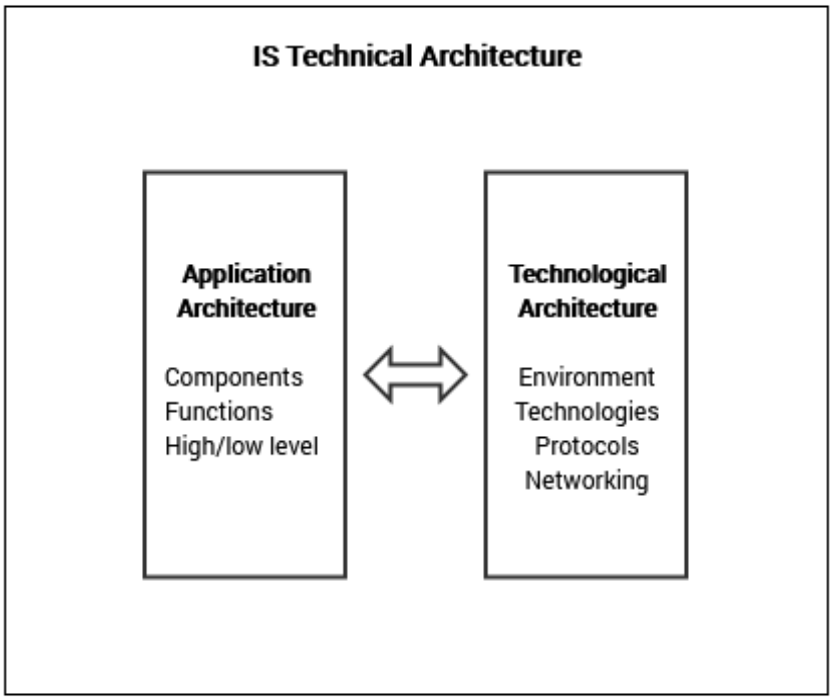

Figure 9. IS technical architecture.

According to Spewak and Hill (1992) application architecture defines the specific applications used in data management and supporting business, and they argue that application architecture should aim to define the functionality of the applications that ensure access to the data, instead of defining the software that was used in the system implementation. Their technological architecture definition concerns the technologies used in building and deploying the applications, such as networks, protocols and computational distribution. (as cited in Vasconcelos, Sousa & Tribolet, 2007.) Zachman and Sowa (1992) use essentially the same principle but offer several layers that model the architecture in increasing detail, depending on the stakeholder. Still, it is debatable how detailed the description should be. IEEE (1999) offers a general guideline that ISA should be high-level. Zijden, Stefan, Goedvolk and Rijsenbrij (2000) argue that ISA is separate from software representation and analysis methods and should focus on the details of the internal system in relation to how they support the business processes (as cited in Vasconcelos, Sousa & Tribolet, 2007).

Although software architecture is generally seen as a separate architectural aggregate, the application architecture portion of IS technical architecture has several common operations, which makes software architecture important to introduce as a concept. According to Kruchten, Obbink, & Stafford (2006), software architecture involves the structure and organization of components and sub-system, which then form a full system. Additionally, "the properties of systems that can best be designed and analyzed at the system level." Aside from this formal definition, they add that software architecture is the designer's documented intention on the system's structure and functionality. As the system becomes increasingly complex, architecture gives a semblance of control over it, throughout its lifecycle. Still, there is no widely accepted definition. According to The International Federation of Information Processing Working Group 2.10 (as cited in Kruchten et al., 2006), there are five subareas related to software architecture. Architectural design governs how the architecture is created. Analysis is about how one can answer questions about the finished products, using the architecture. Realization means how a system is produced based on an architectural

description. Representation is about durable artifacts, that are used to communicate the architecture to different entities in the organization. Finally, economics dictates the architectural issues that drive business decisions. Software architecture is also related to other similar activities, such as software design, system architecture, requirements engineering and quality, to name a few.

The basic high-level building blocks of a software architecture can be summarized as components, connectors and configurations. Component is a computational unit, that has a certain task and logic to adhere to. Connector is a visualization of the relationship between components and the rules governing these relationships. Configuration is the combination of components and connectors in a full system, which describes the structure of the application. (Medvidovic and Taylor, 2010.) Figure 10 visualizes this with a theoretical example.

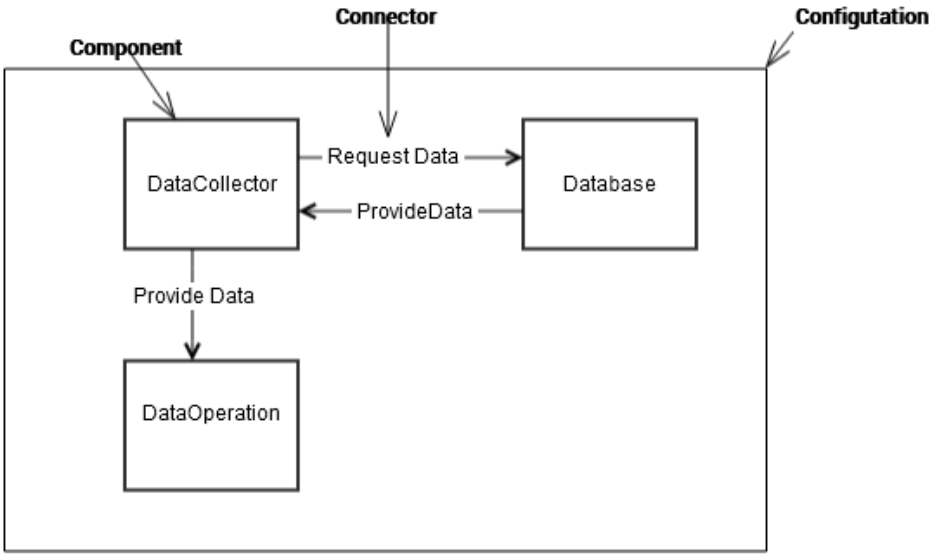

Figure 10. Software architecture notation.

In the example, there are three components, DataCollector, Database and DataOperation, each with some unspecified tasks. DataCollector and Database have connectors going bidirectionally, with DataCollector requesting data from Database and Database providing data to DataCollector. DataCollector has a connector with DataOperation, where DataCollector provides data to DataOperation. The unity of the three is the configuration of this example. The notation in Figure 10 is purely conceptual, meaning that it doesn't adhere to any standard. As Medvidovic and Taylor (2010) note, there are several ways of notation when it comes to describing a software architecture, like how different programming languages have a specific way of describing their operations.

2.5 Virtual Reality

Virtual reality (VR) has a wealth of definitions and terms used to describe it. For example, VR and virtual environments (VE) are often used interchangeably, with added terms such as "synthetic experience", "virtual worlds", "artificial worlds" and "artificial reality" (Mazuryk and Gervautz, 1996). Virtual worlds have also been separately defined to mean persistent and synchronous worlds, where a network of people,

represented as avatars, communicate and coexist. The virtual world is also facilitated by networked computers, meaning that the definition is limited to the digital worlds. (Bell, 2008.) Schroeder (2008) similarly argues that virtual worlds differ from VR and VE in being persistent and the act of socializing with others is the main point. The descriptions of VR are similarly numerous, but they can be summarized to mean an interactive and immersive experience, taking place in a simulated and autonomous world, where the user has a feeling of presence. (Mazuryk and Gervautz, 1996.) Zeltzer (1992) visualized this VR paradigm with what is known as Zeltser's cube (Figure 11).

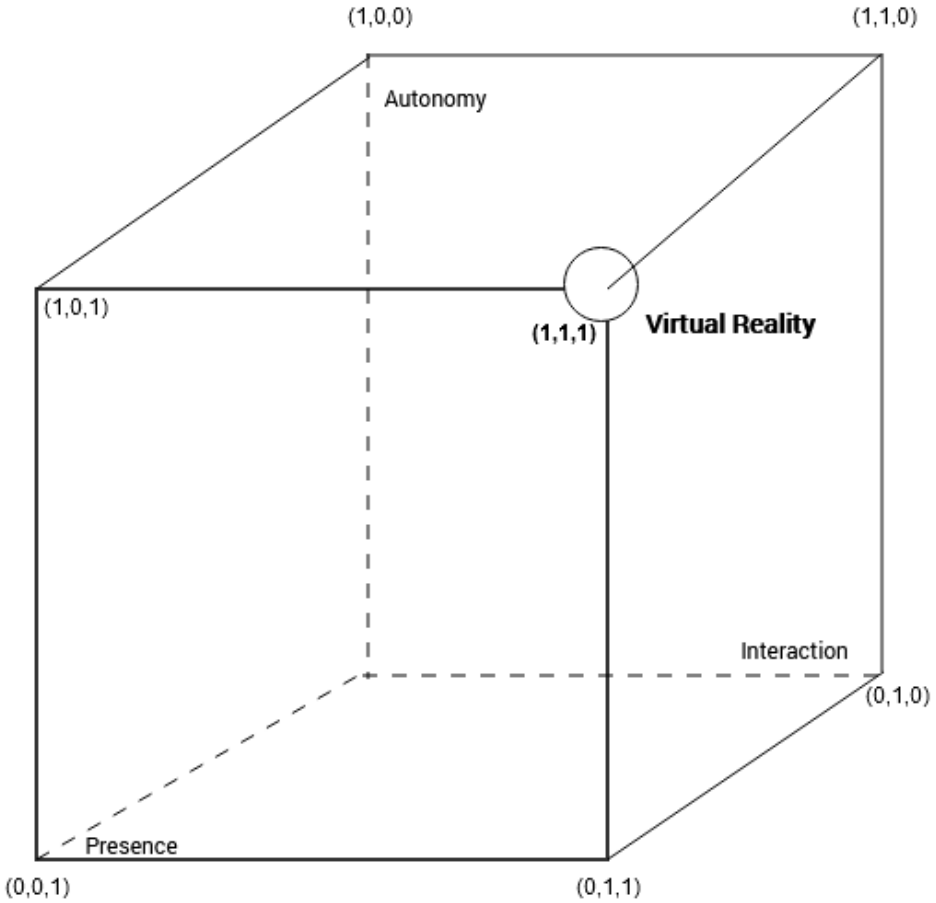

Figure 11. Zeltser's cube (Zeltser, 1992).

The basic idea behind Zeltser's cube is that VR is the combined effort of the feeling of presence, a level of interaction and certain amount of autonomy for the user. Milgram, Takemura, Utsumi & Kishino (1995) present a virtuality continuum, which highlights levels ranging from the real world and the fully virtualized environments (Figure 12).

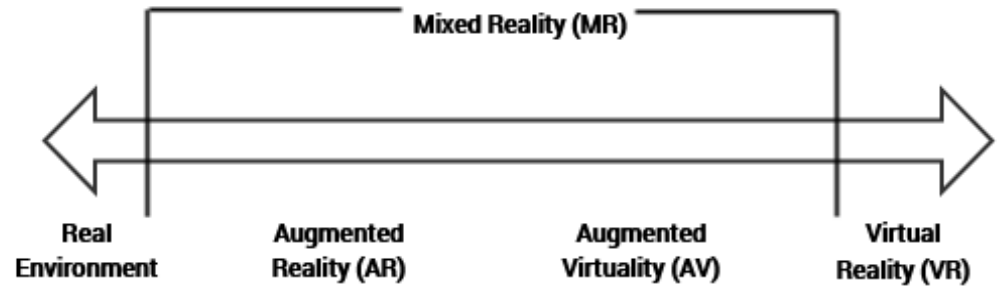

Figure 12. Virtuality Continuum (Milgram et al., 1995).

The purpose of the continuum is to showcase the different levels of relationships that real world depictions and virtual depictions of reality can coexist. The left extreme is some real-world presentation of a real environment, for example a city block. The right extreme would be a completely virtualized presentation of an environment, such as the same city block using 3D models on a computer program. The in-between area of these extremes is what is referred to as Mixed Reality (MR), where reality can be augmented with virtual things or virtual environments can be added on with real objects. (Milgram et al., 1995.) As Ternier, Klemke, Kalz, Van Ulzen & Specht (2012) add, Augmented Reality (AR) applications use the real environment and add virtual elements, while Augmented Virtuality (AV) uses immersive virtual worlds that can be interacted with by using real objects. An example of AR would include a game that displays the real environment on a screen using the device's camera and adds virtual objects around the environment. An example of AV could be a real-world device, that an engineer can control inside a virtual environment.

Slater and Wilbur (1997) consider the technology of VR and VEs as the way to measure its level of immersion. Immersion is divided into inclusive, extensive, surrounding and vivid illusions of reality presented to the user, and the way a computer display can reach these levels determines the immersion. Inclusivity indicates how well the world outside a VE can be filtered out. Extensive means the range of accommodated senses. Surrounding means the level of the field of view available to the user. Vivid means the visual quality of the used displays. Any of these levels, in theory, could be reached by using appropriate technical means. Mazuryk and Gervautz (1996) use immersion as a way to distinctly divide VR into different levels. Desktop VR is a simple application that uses a simple monitor and supports no additional sensory outputs. Fish Tank VR is an improved version, where head-tracking is supported, but conventional monitors are used to convey the image. Finally, when a head-mounted device (HMD) is introduced as a replacement for the monitor, supporting stereoscopic viewing in accordance to user's positioning in the virtual world, we arrive to the modern view of a VR environment. Additions to Mazuryk and Gervautz's definitions include Binocular Omni-Oriented Monitor (BOOM), which is like an HMD, but is handled like binoculars, as well as Audio-Visual Experience Automatic Virtual Environment (CAVE), which a cube of user-facing displays. As the user moves within the CAVE, the correct positional image of the VE appears on the display screens. (Cruz-Neira, Sandin, DeFanti, Kenyon & Hart, 1992.) According to Slater and Wilbur (1997), these definitions could be used to determine the level of "surrounding" in a given VR system, with the lower level definitions of Mazuryk and Gervautz (1996) being toward one extreme and the HMD and CAVE systems toward the other. Figure 13 showcases a typical current HMD-based VR component setup.

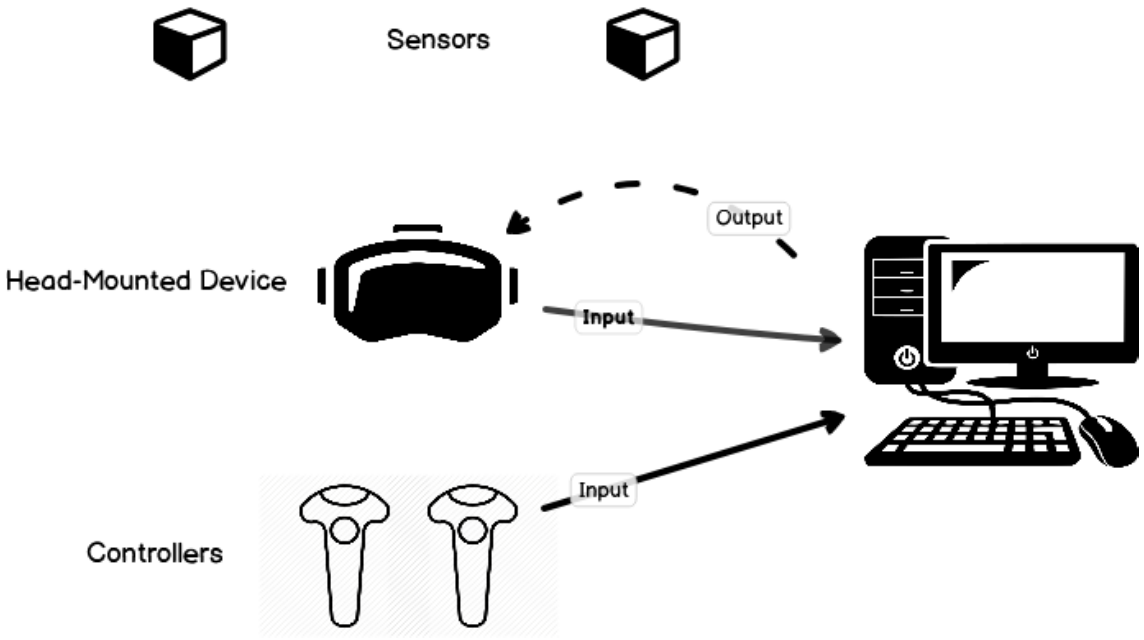

Figure 13. Virtual reality components. Adapted from Mazuryk and Gervautz (1996).

An HMD is the most noticeable of the components and it serves as the main sensory input device, hosting two separate lenses, on which the virtual environment is projected. The HMD also acts as a tracker for the head movement. Controllers are added for giving the user more control of the virtual environment and allowing increased interactivity. A computer is used to as a server that runs the VR application, takes inputs from the user and sends the view of the environment as output to the HMD. (Mazuryk and Gervautz, 1996). An additional component are the sensors, which are typical in today's VR hardware, to create larger areas for usage. The sensors can be utilized to allow for room-scale interactivity, where the user can move freely around the detected area, further increasing the possibilities for interaction.

Applications for VR are similarly numerous, ranging from the entertainment field to industries such as military and manufacturing. Mazuryk and Gervautz (1996) offer a very comprehensive list of VR's possibilities. VR increases potential for data visualization, which can be exploited in fields such as biology and chemistry, as well as engineering. Any design-related practice can be enhanced by VR, as virtual prototyping and planning can give close to accurate results in representing a physical object. Training and education have a long history in VR, especially when it comes to flight-simulators, but medical procedures, such as surgery, can also be modeled digitally. Telepresence and co-operation are other attractive options. The former relates to remote operation of a physical device, like a robot, while the latter references the possibilities of shared virtual environments, where various tasks can be solved by multiple people simultaneously. Earnshaw (2014) has likewise collected a plethora of ideas related to VR implementation, such a virtualized laboratory facility or using VR in animation process.

In summary, IS is the collection of networked components that provide value for the end-users and the business that implements it. When creating an ISA, EA frameworks, such as Zachman's framework, TOGAF and FEA, have been used as a baseline. In general, ISA consists of data architecture, application architecture and technological architecture. Data architecture consists of what data the organization uses and needs, and the ways data is managed and stored. Application architecture is an offset of software architecture and aims to provide an overview of the functions of the applications that consist the IS. Technological architecture presents the technological

components and methods, such as messaging protocols, networks and hardware. The final concept defined is VR, which can be defined as a virtualized experience of reality, where presence, autonomy and interaction come together. Typical VR experience consists of using an HMD and controllers, which feed inputs to a host computer, which then relays output to the HMD in the form of a representation of the virtual environment.

3. Research Method

The research problem of this thesis is finding a solution for an information system, that makes it possible to combine organizational actors and assets in a VR environment, where various users can co-operate. The research will be conducted as a part of a small team at the target organization. The research method selected to solve this research problem is Design science research (DSR). DSR constitutes the study of the artificial, the man-made objects and artifacts. This chapter introduces DSR as a concept, specifies its use in IS research and presents how this research fulfils the research method.

3.1 Design science research

Simon (1996) dubs design science as "science of the artificial", meaning that it concerns itself with the human created objects and phenomena. According to Vaishnavi and Kuechler (2004), DSR aims at creating new knowledge by designing novel and innovative artifacts, such as thing and processes, and analysing the use of the artifact with reflection and abstraction. Some examples of artifact types include algorithms, interfaces, system design methodologies and languages.

While many versions of the DSR process models exist, they share similar steps with one another. Figure 14 showcases one model of the DSR process. (Vaishnavi and Kuechler, 2004.)

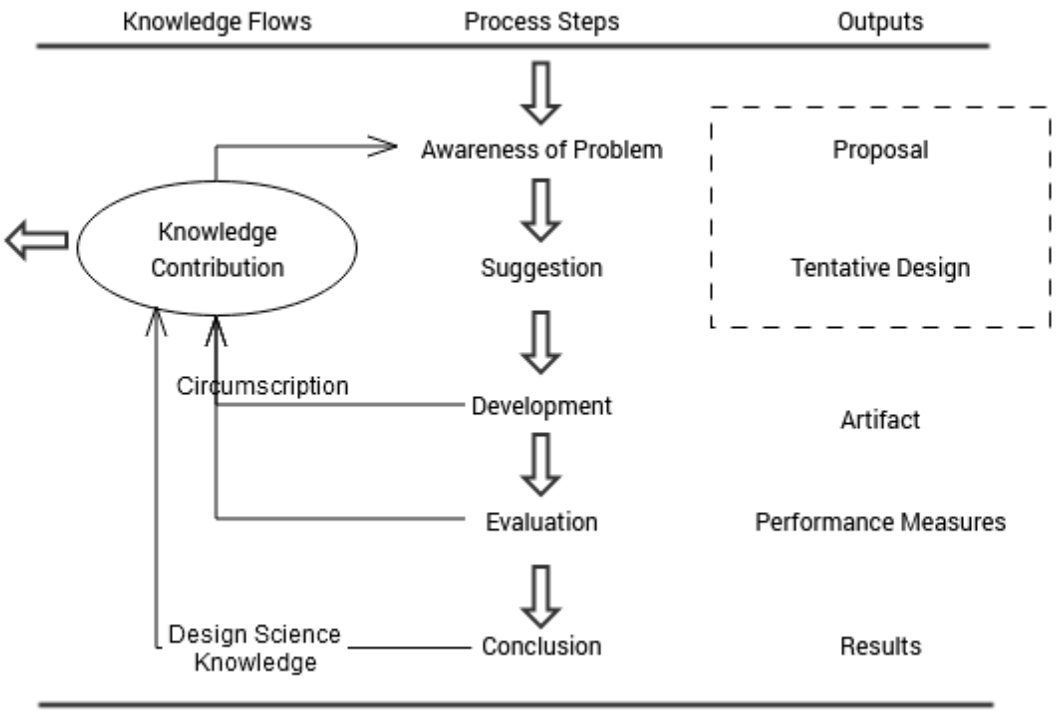

Figure 14. Design science research process model (Vaishnavi and Kuechler, 2004).

The process starts with gaining awareness of the problem, which can come from new developments in the related industry, or from transferring problems and solutions from other fields. Awareness is followed by the proposal, where a new research effort is briefly outlined. Proposal further evolves into a suggestion and a tentative design, which demonstrates at least some of the intended functionalities of the design. If the entity interested in conducting a DSR about the topic cannot create a tentative design, the idea is scrapped for the time being. The dotted line around proposal and tentative design represents the close relationship that these two phases share. Once the tentative design is accepted, development begins in earnest, where the target artifact is further designed and then implemented. Implementation is entirely dependent on the artifact and the implementation by itself doesn't have to employ novel building blocks, as the design itself should primarily carry the novelty. After the artifact is constructed, it is evaluated based on relevant criteria. These criteria are always implicit and frequently made explicit early in the process. The evaluation phase feeds the process of DSR with further hypotheses and suggestions for new designs. (Vaishnavi and Kuechler, 2004.) Evaluation criteria isn't entirely agreed upon subject. Hevner, March, Park & Ram (2004) note how the utility is the most typical focus, but Gill and Hevner (2013) suggest the artifact's adaptability and survivability in its environment as a possible viewpoint. The final phase of the process is conclusion, where the results are documented, the design judged, and the knowledge gained categorized as either firm or such that contains loose ends, depending on the repeatability of the gathered facts and the possible anomalous results that defy explanation. (Vaishnavi and Kuechler, 2004.)

As information systems concern many different areas, such as IT strategy, organizational structure and IS infrastructure, DSR must be adapted to fit them. Figure 15 presents a framework for utilizing DSR in IS research.

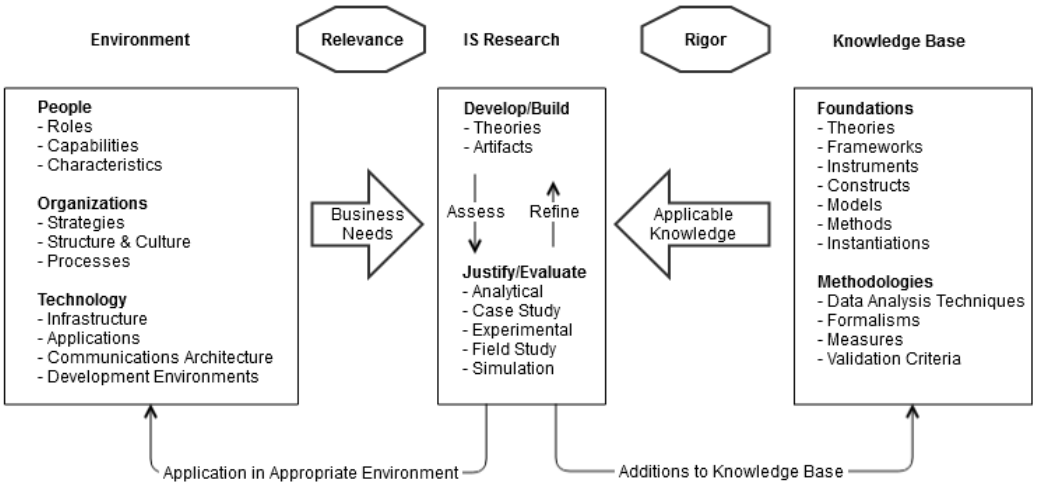

Figure 15. DSR in IS research (Hevner et al., 2008).

The environment is the setting for the IS research. The environment includes the people, the organizations and the technologies available. These are combined, and the problem area thus identified and addressing business needs with the research activities assures relevance of the research. IS research happens in two phases, development and evaluation, where new solutions are created and then judged based on criteria. Applicable knowledge is gathered from the knowledge base, which include theories, frameworks, instruments, constructs, models, methods, instantiations, data analysis techniques, formalisms, measures and validation criteria. DSR should be differentiated from routine design. For example, solving an organizational problem by creating a

financial information system, using existing best practices. DSR emphasizes both solving new problems, finding new ways to solve old ones or applying existing solutions in new areas. Additionally, contributing to the knowledge base is a vital part of DSR, something routine design in a typical organizational development doesn't include. (Hevner et al., 2008.)

3.2 Seven guidelines for DSR

Table 2 presents seven guidelines for conducting DSR. While these constitute important parts of the research effort, they shouldn't be taken as absolutes. It is important for the researcher to use their own judgment and expertise in deciding how and when to follow each guideline. (Hevner et al., 2008.)

Table 2. DSR guidelines (Hevner et al., 2008).

Guideline	Description
Guideline 1: Design as Artifact	Must produce viable artifact in the form of a construct, model, method or instantiation.
Guideline 2: Problem Relevance	Develop technology-based solutions to relevant and important business problems.
Guideline 3: Design Evaluation	Utility, quality and efficacy must be rigorously evaluated using well-executed evaluation methods.
Guideline 4: Research Contribution	Clear and verifiable contribution in the areas of design artifact, design foundations and/or design methodologies.
Guideline 5: Research Rigor	Application of rigorous methods in construct and design of the artifact.
Guideline 6: Design as a Search Process	Utilize available means to reach desired ends in search of effective solutions.
Guideline 7: Communication of Research	Must be presented effectively both to technology-oriented and management-oriented.

Artifacts are the main concrete deliverable of the DSR effort, but the definition of the artifact itself is not entirely agreed upon. It can be argued that the artifact should be a combination of the IT solution and its environment (Orlikowski and Iacono, 2000; Weber, 2003), but there are also views that separate the two areas. Hevner et al. (2008) distinguish the artifacts as constructs, models, methods and instantiations, which encompasses the language of the artifact and the demonstration of its practical feasibility.

DSR aims at creating innovative solutions, to problems that have a real-world need to be solved. One definition for a problem is the difference between the states of the current situation and the goal and the activity of solving the problem is the search for solutions that decrease these differences. The problem imposes certain restrictions or requirements on the system, either implicitly or explicitly, and in business situations, the goal is typically to increase profit or decrease costs. Problem relevance is also relevant to the community, which in IS research constitutes all those entities and organizations that use IT to create more efficient ways of working. New knowledge in this area is what all these practitioners are after. (Hevner et al., 2008).

The artifact must be evaluated rigorously, using appropriate methods. The methods are typically based on already existing knowledge base, which are summarized in Table 3 (Hevner et al, 2008).

Table 3. DSR evaluation methods (Hevner et al., 2008).

1. Observational	Case Study: Study artifact in business environment. Field Study: Monitor use of artifact in multiple projects.
2. Analytical	Static Analysis: Examine structure of artifact for static qualities. Architecture Analysis: Study fit of artifact into technical IS architecture. Optimization: Demonstrate inherent optimal properties of artifact or provide optimality bounds on artifact behavior. Dynamic Analysis: Study artifact in use for dynamics qualities
3. Experimental	Controlled Experiment: Study artifact in controlled environment for qualities Simulation: Execute artifact with artificial data
4. Testing	Functional (Black Box) Testing: Execute artifact interfaces to discover failures and identify defects. Structural (White Box) Testing: Perform coverage testing of some metric in the artifact implementation
5. Descriptive	Informed Argument: Use information from the knowledge base to build a convincing argument for the artifact's utility Scenarios: Construct detailed scenarios around the artifact to demonstrate its utility

As with the guidelines, the evaluation methods must be selected such that they produce appropriate results. Usually certain artifacts have best practices in terms of evaluation, and deviation should be used with caution, in situations where a common method is either unfeasible or the artifact is so innovative that it cannot be judged with any typical way. The methods themselves can be slotted into observation, analysis, experiments, tests and descriptive evaluations. Observation are studies about the artifact's use in an environment, either in a real use case or artificially in a lab. Analytics utilize separate established methods or patterns that can be juxtaposed with the created artifact and how it functions or fits the intended purpose. Experiments can be controlled, where certain qualities are studied, or simulations, where artificial data is used to execute the artifact operations. Testing can be done by executing artifact interfaces to find failures, or to decide on certain metrics used in a coverage test. Finally, descriptive evaluation are theoretical demonstrations and arguments for artifact's utility, based on prior research and justified imaginary scenarios. Like evaluation of the artifact, the research itself must

be rigorously implemented and carried out. This is another aspect where the skill of the researcher has a major role, as too much rigor can decrease the relevance of the research, while too little rigor can jeopardize the value of any possible solution. While DSR often relies of mathematical formalism, IT artifacts aren't always such that fit into formulas. Rigor can still be applied to the artifact's applicability and generalizability, as well as in selecting appropriate evaluation methods. (Hevner et al., 2008.)

The research contributions can be divided into design artifacts, foundations and methodologies. The design artifact is the most typical contribution of DSR. In IS research, design methodologies, design tools and prototype systems are most common artifacts. Foundations add directly to the knowledge base through their innovative nature. Some examples include problem and solution representations, design algorithms and innovative information systems. The last contribution category is methodologies, which mean ways of evaluating the IT artifact. One way for these to manifest are evaluation frameworks. Whatever the contribution, they must be implementable, and they most solve an important, novel business problem. (Hevner et al., 2008.) As Vaishnavi and Kuechler (2004) write, the design science knowledge is the main goal when conducting DSR. Gregor and Hevner (2013) visualize this by using a knowledge contribution framework (Figure 16).

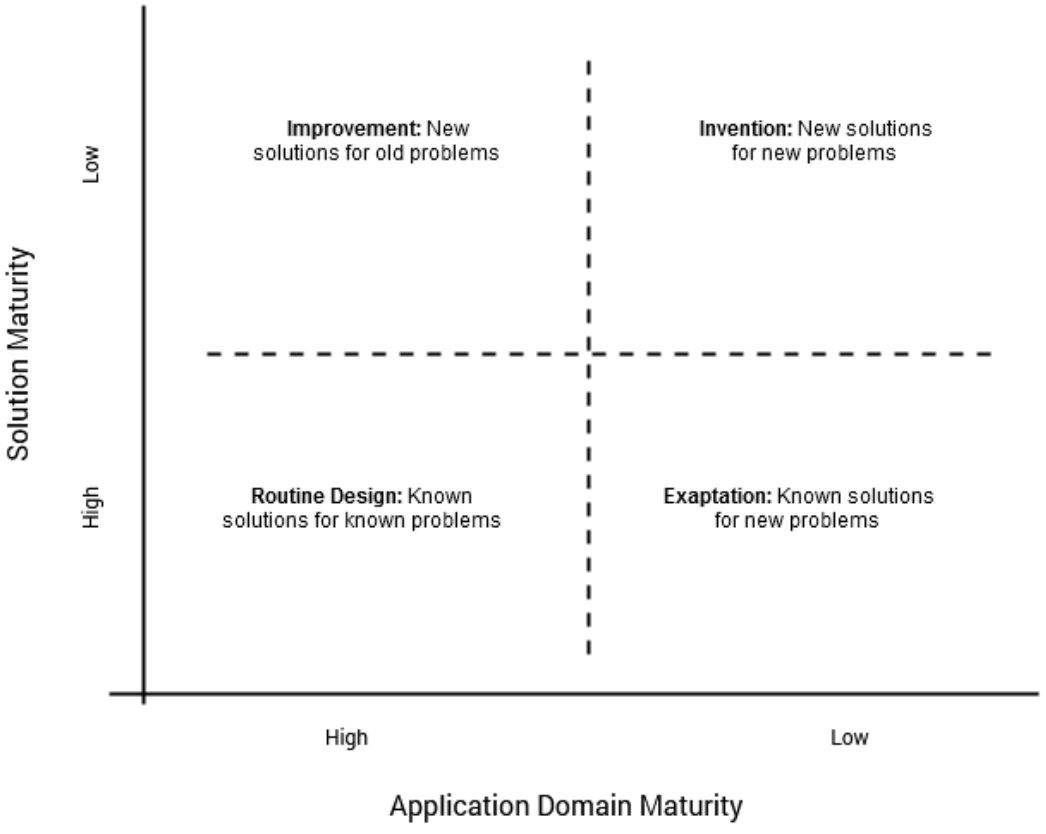

Figure 16. DSR knowledge contribution framework (Gregor and Hevner, 2013).

The lowest level of the framework is the routine design when existing solutions are applied to existing problem (Gregor and Hevner, 2013). While this doesn't ordinarily constitute as DSR, routine design can sometimes lead to it, by uncovering new problem areas (Vaishnavi & Kuechler, 2004). The improvement area aims at creating a design that is demonstrably more efficient and effective in its intended purpose, than an already existing design. Researcher must therefore have a close understanding of why the

current solutions don't either exist or are suboptimal. The opposite contribution of improvement is exaptation, where known solutions are used to solve new problems. These types of contributions are typical in IS, since as new technologies are created, they often require new applications and systems to be built for their use case. Exaptation research should be interesting and non-trivial and show the value of extending the known solutions to the new problem. The highest level of DSR is invention, where breakthroughs happen through new solutions to new problems. The process of inventing can be described as a creative exploratory process through multiple complex problem areas, and it becomes DSR when it results in an artifact that can be evaluated, and it contributes new knowledge. In this area, no effective artifacts exist, and the research questions may not have been formulated before. (Gregor and Hevner, 2013.)

Design in DSR is a search process, the goal of which is to find the optimal ways to solve the given problem. The problem can be divided into sub-problems, which act as a starting point, and as new layers are discovered, the search process works iteratively. In IS research, the difficulty comes from determining so-called standard or general means and ends to a problem, as the field is by nature very chaotic and filled with variables. First trying to figure out a list of all possibilities, before choosing the best one, is infeasible. Therefore, the typical approach is to look for satisfactory solutions, that work well-enough in a given set of restrictions and requirements. It is important to have a solution that works, even if explaining the reasons why it works cannot be readily explained, as the demonstration of applicability opens new avenues for later research and knowledge contribution. (Hevner et al., 2008.)

DSR should be communicated with technical and management personnel in-mind. The technology-oriented personnel require such a representation, that the artifact can be implemented in a suitable setting, allowing for further implementations, iterations and extensions. The management wants an understanding of how implementing the artifact affects the organization, for example, on a financial level. The focus should therefore be on the knowledge of how the artifact should be implemented on a high-level. (Hevner et al., 2008.)

3.3 DSR in context

The model of Figure 14 can be adapted to give an overview of the process for the research of this thesis. Table 4 describes the tentative steps in this context.

Table 4. DSR process steps for this thesis' research.

Process Step	Context in thesis research
1. Awareness of problem	Possibilities of digitalization and virtualization of the target organization's business processes.
2. Suggestion	An IS based on asset sharing and virtual environments, where co-operation is utilized.
3. Development	Architectural specification for a POC environment.
4. Evaluation	Implementation of the POC architecture. Other methods as deemed fit.
5. Conclusion	Results depending on the outcomes of the study.
Contribution	A novel IS based on virtual reality.

The research problem comes from the interest that the target organization, has in exploring digitalization and VR as a possibility in bringing some of their business processes to the digital world. VR is seen as a way to decrease costs in areas such as prototyping and co-operation between intra- and inter-organizational partners. The suggestion to solve this problem is an IS that combines asset sharing and virtual environments, in a way that allows the relevant stakeholders to work together in virtual environments mirroring physical locations and processes related to the target organization. The development phase consists of creating an architecture specification, that allows for a POC implementation, while also offering insight into the possibilities and restrictions going forward. Evaluation will be mostly conducted by implementing the proof-of-concept architecture into a working system. Other methods for are used as the situation requires. Using Table 3 to map out the exact evaluation methods, the implementation falls into case study or field study, depending on the situation.

The contribution of the research is a novel ISA, using existing methods. Using the contribution framework presented in Figure 16, this research's contribution falls into the Exaptation-category (Figure 17).

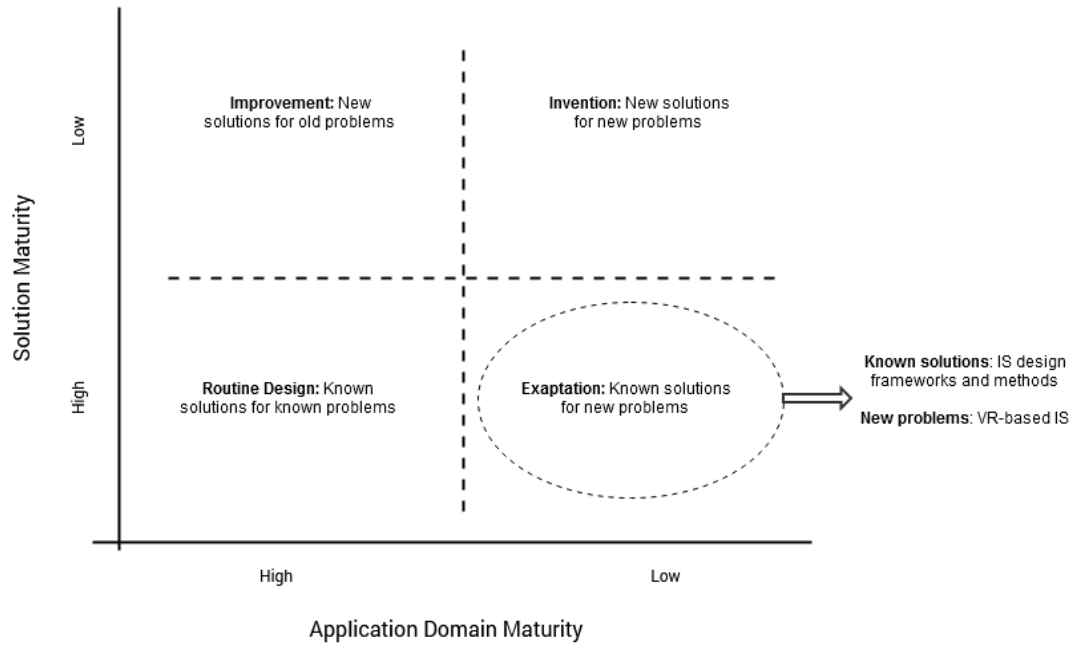

Figure 17. Contribution of this thesis, in the context of Gregor and Hevner's (2013) model.

Using Gregor and Hevner's (2013) definition of the Exaptation-contribution, this research's contribution combines known solutions found in IS research and development, to solve a new problem, which is a VR-based digital IS. The main contribution will the specification of the IS architecture, while the implementation will serve as proof of its viability and grounds for further research and development.

4. Overview and Requirements Engineering

This chapter outlines the overview of the developed IS, referred to as *Saturn*, and the motivation behind it. Afterwards, the requirements for the POC system are developed and described.

4.1.1 The Idea of Project Saturn

The target organization specializes in data networking and telecommunications equipment, which they provide for their customers worldwide. The common business practices currently exist in analogue forms, with iterative physical designing and prototyping of products and location-based meetings with customers. There are several entities that take part in these processes, such as designers, factory workers, sales personnel and presenters at various exhibitions.

Saturn comes from the wish to digitalize these business processes and to utilize VR as a platform for reducing costs and providing possibilities that the physical world limits. As it stands, the business processes of the target organization are restricted and hampered by cost and logistics. Physical prototyping of products is slow and expensive, but necessary when customers come up with new requirements or when existing products don't match their needs. Additionally, having to physically travel to different locations for presentations, testing and sales events are an added cost.

The idea behind *Saturn* is a co-operative VR environment, where tasks such as prototyping, testing and presenting can be conducted regardless of geophysical location. The system would emulate various physical locations, where users could visit and do various tasks, such as familiarize with telecommunications products, visit laboratories and factories and learn about networks. Using specific virtual views, the users of the system can get, for example, information about networks and understand how to prepare for future expansions. Various simulation results are utilized to increase relevance of the things visualized in VR.

To supplement the VR application, a website would be created for the system. On the website, the user would find a download link for the client application, that is used to run the application. The website would also be hosting an Asset Store, where developers can store, exchange and comment on assets that are used in creating the virtual environments.

4.1.2 Requirements

The requirements engineering was started with this final concept in mind. The first iteration consists of mapping out the relevant requirement areas and recognizing aspects that the final system needs to meet. The requirements are first divided into functional and non-functional requirements. As Adams (2015) writes, functional requirements describe the functions that the system should be able to realize. In other words, functional requirements tell what the system does. The functional requirements can be divided into tasks each stakeholder wants to be able to carry out. The first iteration of functional requirements was creating simple use case diagrams for what a generic user

and a generic developer would want to accomplish in a finished version of the system. Figure 18 shows user actions on the website.

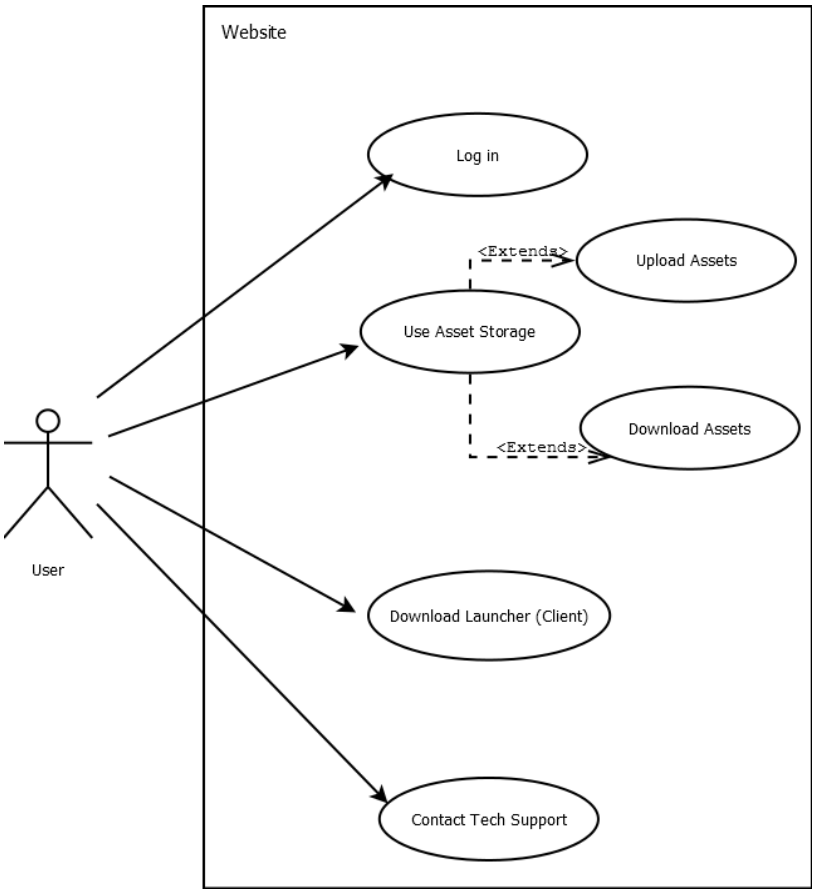

Figure 18. Use Case 1: User uses the website.

The two main purposes of the *Saturn* website are the client application (launcher) download and the asset storage. Launcher can be downloaded by anyone and the user credentials mandate the level of access rights. Asset storage is only used by developers. The asset storage can be used for uploading and downloading assets used to create virtual environments. The website also offers some level of technical support, such as how to get started with using the app.

Figure 19 shows user tasks on the downloadable client.

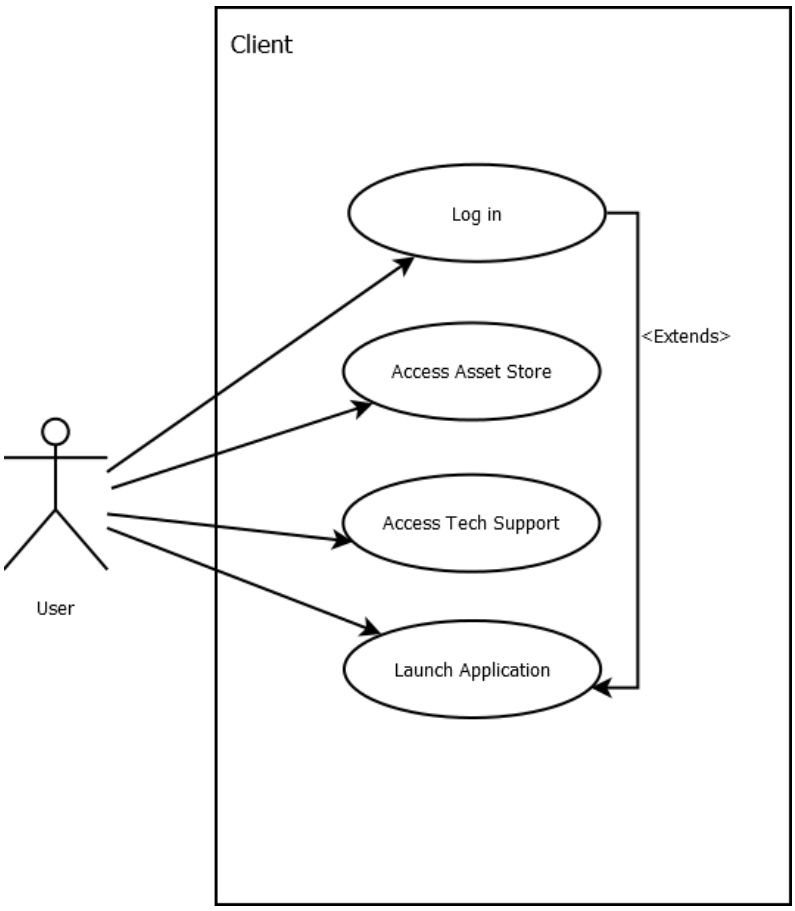

Figure 19. Use Case 2: User uses the client software.

The client software's main purpose is launching the application. The user first logs in, and then gains the ability to launch the VR app. Available VR content depends on the access rights that the user has, based on their role. Client also has links, that take the user to the website, either to the asset storage or tech support. This gives the user an easier access between the website and the app itself.

Figure 20 shows developer tasks in the *Saturn* system.

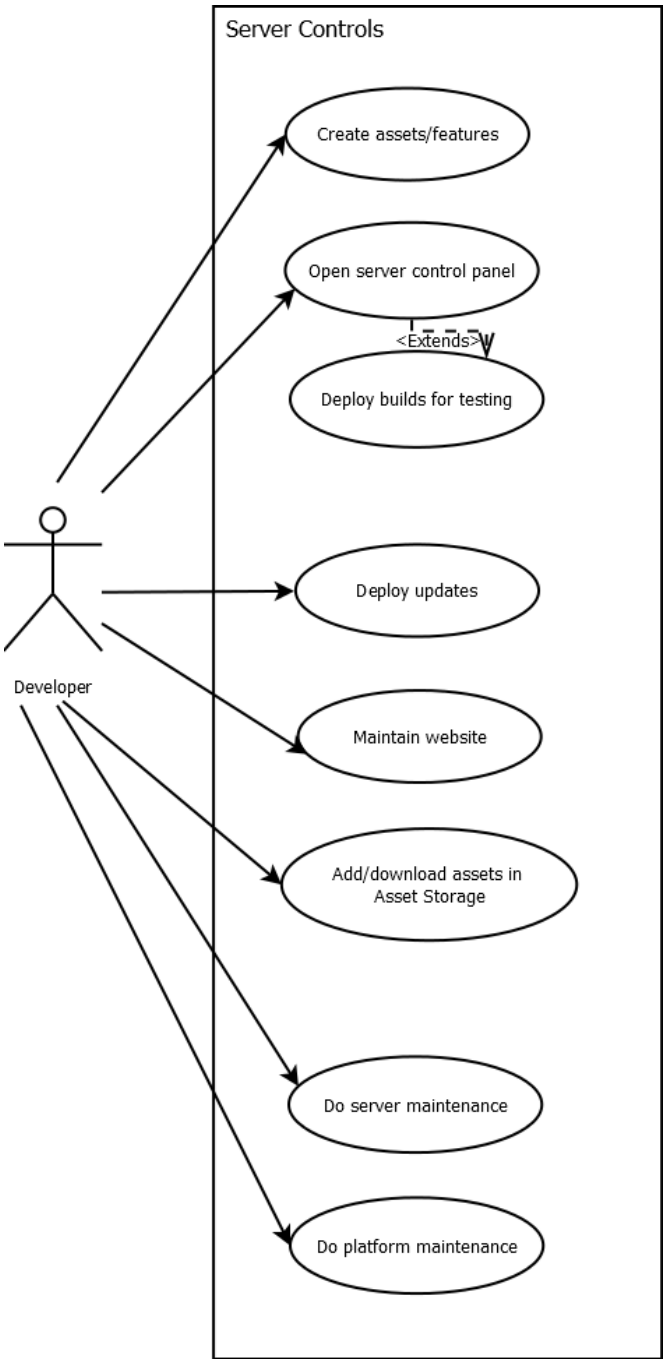

Figure 20. Use Case 4: Developer tasks.

Developers can include many roles within the organization, such as designers and programmers. Asset creation can mean 3D product models, VR assets or scripts, and they can be added and downloaded in the asset storage. Additionally, designated developers can deploy new builds of the VR app for testing and eventually to end-users. The server environment can also be maintained, both in terms of the servers for separate components of the system and the platform that hosts the servers.

Finally, the VR application should allow for co-operative use of specified tools in various VEs. Depending on their level of access, the user could engage with processes that can be equated with the target organization's operations today, in some form.

From these four groups of use cases, the second iteration of functional requirements were gathered, fitted for the smaller scale POC implementation. Table 5 consists of the selected requirements.

Table 5. POC functional requirements.

ID	Functional Requirement
FR1	Website allows for user login
FR2	Website has a downloadable client software
FR3	Website has some level of technical support
FR4	Website has some implementation of asset storage
FR5	Client software can be logged into
FR6	Client software can be used to launch the VR app
FR6	VR app has some locations the user can visit and move around in
FR7	VR app has some products that can be interacted with
FR8	VR app allows for co-operation
FR9	VR app can be updated
FR10	New builds of the VR app can be deployed
FR11	Maintenance tasks can be done for the website and the servers

For the website, the requirements were deemed to mostly fall in line with the original ideas, just implemented in a smaller-scale. Asset storage was the most ambiguous requirement, and its planned form and finalized functions weren't clear. Client software was planned to be as simple as possible for the POC environment, with a simple login screen and app launch functionality. Considering the POC would be in restricted use, refining the client wasn't deemed necessary. The restrictions set by the available resources had to be considered. Therefore, the requirements were set for having certain environments that can be visited co-operatively and have interactive products in them. Maintenance tasks were mostly the same as the original use cases showed, at least in a larger scale, with the possibility for maintaining the servers, website and the app itself.

Non-functional requirements describe how the system should carry out the functional requirements. They define the behaviour of the system. Non-functional requirements are further divided into sub-categories. As Adams (2015) writes, non-functional requirements cover a the characteristics of a system. Chung, Nixon, Yu and Mylopoulos (2012) add that noon-functional requirements describe how a system completes a given function, instead of describing what the system does. Table 6 shows the ranking of non-functional requirement categories in a quality grid.

Table 6. Quality grid for non-functional requirements.

Quality Factors	Critical	Important	As Usual	Unimportant	Ignore
Operation					
Security	x				
Data Integrity		x			
Reliability/Availability	x				
Usability		x			
Performance	x				
Revision					
Maintainability			x		
Testability			x		
Adaptability/Extensibility			x		
Transition					
Portability				x	
Interoperability				x	
Reusability			x		

The quality grid shows the ranking of quality attributes on a five-point scale, from *critical* to *ignore,* depending on the perceived importance on the system. Security, data integrity and reliability/availability were deemed to be critical for the finished system. Security is critical, because the system consists of business-critical material. Data integrity is critical, because the system is used in sales and promotion, and the customers need to be able to trust that the information is correct. Reliability and availability are critical, because the system is designed to be always online, and reachable regardless of time and place. Finally, usability and maintainability were deemed as the final higher-than-average level requirements. Usability is important, because VR is not a widely adopted technology and increasing the chances of its use rely on the system being easy to use. Performance is important for the same reasons as usability and to decrease the chance of health issues during the VR use, the performance should be a high priority. These VR related health issues are also called "cybersickness", which can result in similar symptoms as in cases of motion sickness, including nausea, headaches and dizziness (Rebenitsch and Owen, 2016).

Using this quality grid, a list of non-functional (quality) requirements was derived. As the use case scenarios, which were derived from the initial vision of the system, the

quality requirements also started from a bigger view than the strict POC system. The first iteration is listed on Table 7.

Table 7. The first iteration of quality requirements.

ID	Quality Requirement
Quality	
QR1	System must scale to support a certain number of users
QR2	Fast response time
QR3	High framerate
QR4	Localization for different regions
QR5	The data handled by the system must be up to date and correct
QR6	Downtime to be minimized when servers are in maintenance
QR7	Automatic error messages and reports
QR9	Always online
Security	
QR10	Encrypted connections
QR11	Encrypted databases
QR12	Access control on a general sense
Modifiability	
QR13	Servers updatable
QR14	Additional servers for backup
Usability	
QR15	Designs in the VR aimed toward helping usability
Environmental	
QR16	Region based servers
Compatibility	
QR17	Different operating systems

The first iteration of non-functional requirements consists of six areas derived from the quality grid, these being quality, security, modifiability, environmental and compatibility requirements. The quality requirements are such that don't necessarily fit

into any other category and signify some inherent quality that the system is envisioned to reach. Some of these include performance metrics, such as high framerate and fast response time, which are necessary for ensuring a comfortable user experience. As the plan for the system is to be co-operative, it should scale to support a certain number of simultaneous users. Since the target organization is a global company, localizing some of the content may be a necessity. This ties in with the global and constant availability. The data that the system uses should be correct, as it is to be used in business situations.

Considering the small-scale deployment of POC, the databases were not expected to contain a lot of sensitive information. Access control would be implemented in a simple state, meaning that there would be some way of controlling, what the users have access to in the system and in the VR app. Finally, usability would be tackled in having system deliver automated error messages and having the VR environments be designed with at least a rudimentary usability issues in mind. While the focus was to build a working POC, some fundamental usability ideas would be used when applicable.

The system should be modifiable in terms of allowing server updates and having backups for redundancy. As VR-based systems are not very common, the VR environments and functions should be designed with usability in mind.

After this first-pass, another round of iteration was conducted, with the aim of specifying some requirements to a lower level, while also removing requirements, which were deemed to be not in the scope of the POC implementation. The second iteration is shown in Table 8.

Table 8. The quality requirements for POC system – second iteration.

ID	Quality Requirement
QR1	Multiple users must be able to interact in the same environment
QR2	Response time of below 200 milliseconds
QR3	Framerate of 90 frames per second
QR4	The data handled by the system must be up to date and correct
QR5	Downtime to be minimized when servers are in maintenance
QR6	Automatic error messages and reports
QR8	Always online
QR9	Encrypted connections
QR10	Encrypted databases
QR11	Access control on a general sense
QR12	Servers updatable
QR13	Additional servers for backup
QR14	Designs in the VR aimed toward helping usability

Performance metrics in the second iteration were focused on some agreed-upon values, in this case, 200ms for delay and 90 frames per second for framerate. Response time needs to be high, for significant delay between the user giving an input and getting an output to the HMD can cause effects of cybersickness (Rebenitsch and Owen, 2016). Framerate has the same adverse effect if it doesn't reach high-enough levels. The scaling of the simultaneous user-amount was left undefined, as the only vision was eventual co-operative experience. Therefore, the POC should be able to demonstrate at least more than one person in a given environment. Data integrity is also eventually an important factor. Therefore, any product-related data that is planned to be expressed realistically, should appear so in VR.

Continuous availability was another contention of the first iteration of quality requirements. While it wasn't clear exactly what kind of server environment was going to be available for usage, the POC would at least have the goal of being always online and have minimized downtime. According to Bell's (2008) definition, this would make *Saturn* a type of virtual world, as persistency and synchronous co-existence of users would be an eventual goal. Redundancy in equipment would be another goal to reach this state and having a way to update the servers if possible. Connectivity could be tested using other target organization sites. While connectivity and persistent availability were deemed important, requirements related to localization were removed in the second iteration, as they were deemed to be out of scope.

Using Table 7 and Table 8, the final list of requirements was gathered. This final iteration was done to have a single collection of requirements, as well as to have a final acceptable list of functional and non-functional requirements. The final list is gathered in Table 9.

Table 9. The final list of requirements for the POC system.

ID	Requirement
FR1	Website allows for user login
FR2	Website has a downloadable client software
FR3	Website has some level of technical support
FR4	Website has some implementation of asset storage
FR5	Client software can be logged into
FR6	Client software can be used to launch the VR app
FR7	VR app has some locations the user can visit and move around in
FR8	VR app has some products that can be interacted with
FR9	VR app allows for co-operation
FR10	VR app can be updated
FR11	New builds of the VR app can be deployed
FR12	Maintenance tasks can be done for the website and the servers
QR1	Multiple users must be able to interact in the same environment
QR2	Response time of below 200 milliseconds
QR3	Framerate of 90 frames per second
QR5	Downtime to be minimized when servers are in maintenance
QR6	Automatic error messages and reports
QR8	Always online
QR9	Encrypted connections
QR10	Encrypted databases
QR11	Access control on a general sense
QR12	Servers updatable
QR13	Additional servers for backup
QR14	Designs in the VR aimed toward helping usability

There were no changes deemed necessary for the functional requirements, that were derived from the original use case diagrams. The only quality requirement that was

removed in the final iteration was the requirement for data correctness. While it is eventually an important requirement, the POC system would only see limited audience, mostly with other developers, so it wasn't deemed necessary to impose a requirement for it at this stage.

5. Architecture Design

This chapter outlines the design of the POC architecture. Using the prior research, and the ISA frameworks as a baseline, the architecture is divided into data-, technological- and application architectures.

5.1 Data Architecture

Data architecture of *Saturn* can be divided into two separate areas. The first area is focused on the creation of material for the VR environments, while the other is focused on the data that users and technical environments use.

5.1.1 Asset Creation Process

Virtual environments require many forms of data in their creation process. The first step in the data lifecycle is identifying the data that is required. Table 10 shows the data used by the POC VR application.

Table 10. Asset data forms.

Data	Description
3D	
Product models	3D models of organization's products that are usable in the VEs.
Environments	Environmental data that is the basis for VEs.
Miscellaneous 3D objects	Either other interactive objects or non-interactive ones for immersion.
Object Animations	Animated elements and objects add functionality and real-world presentability.
2D	
2D visual objects	Graphics for immersion and presentation.
Text	
Scripts (code)	Behaviour for entities in the virtual environments.
Product behaviour	Data from simulations and recordings.
Product specifications	Data sheets that can be used for reference.
Other	
Audio	Sound effects to aid usability and immersion.

The immediately visual data categories can be divided into 3D and 2D assets. 3D encompasses the 3D objects that the VEs are built from. Product models are the most crucial in terms of business operations, as they are the focus of the *Saturn* concept. Environments are the VEs themselves, which can be hand-crafted or based on map data. Miscellaneous 3D objects are any other 3D assets that aren't products or directly related to the environments. For example, tools that can be used for interactions. Animations can be used to demonstrate the functionalities of products, as well as add to the presentation by having realistic movements. 2D objects are essentially graphics that make the VE into more believable and presentable. They are most typically used to give the various 3D objects their intended look. 2D graphics typically manifest as textures and materials. A texture can be any kind of graphic, that is used to "paint" a 3D object to its intended visual look, such as an imported image file. A material, on the other hand, is a collection of data values, and can include, for example, several textures or other data such as scalar or vector data. The result can be used to give an object an appearance of a real-world material, such as wood or metal.

The non-visual data exist in textual and audio forms. Scripting is the programmed logic that gives all the entities in the VEs their behaviour. Scripts are inherently tied to the creation of VR experiences, as without them, there would be no functionality to interact

with. Product-related data can be used for direct functionality and for reference. If the intention is to showcase a product's capabilities using simulations and recordings, this data needs to be gathered. Typically, this happens in textual form. Product specifications can be any other data that a user might want to know about the product. Whereas product behaviour can be visualized, product specifications can be simple data sheets that are used as a reference. The final form of data is audio, which is another way to add to the usability and immersion, as hearing is another way to ease the use of VR objects.

Once these data forms are identified, another task is to determine the source for this data. This gives insight to who is creating what and what ways can it be brought to the *Saturn* environment. Table 11 show this identification.

Table 11. Data sources.

Data	Source
Product models	Internal Designers
Environments	Internal Designers External Developers
Miscellaneous 3D objects	Internal Designers
Animations	Internal Designers
2D visual objects	Internal Designers
Scripts (code)	Internal Developers External Developers
Product behaviour	Internal Labs
Product specifications	Internal Labs
Audio	Internal Designers External Designers

While most of the sources are internal, the logistical fragmentation can be significant. Product modelling can be tasked to mechanical designers, while environments and other 3D objects are created by a core VR development team. This same team can be responsible for scripting and overall implementation. Product related data typically comes from the labs, where physical products are tested and simulated. Other designs within the organization may be from other sources yet.

External sources can vary from partnered vendors to open source or commercial assets. The important thing to note in these situations is how the data is gathered and how the organization can guarantee a level of control of the data. Areas such as licensing need to be considered.

Once the initial data identification is complete, the issue of the data lifecycle can be designed. The lifecycle of the VR asset data is visualized in Figure 21.

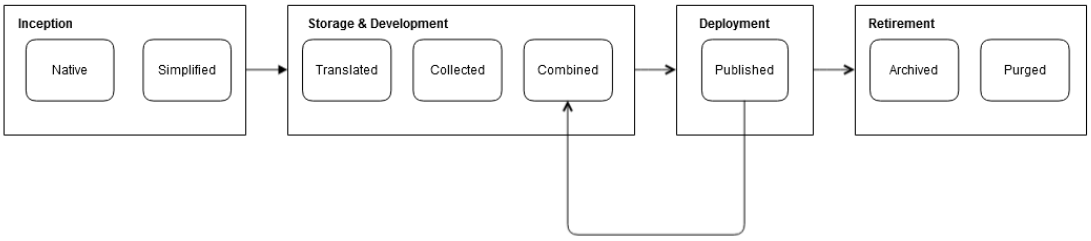

Figure 21. VR asset data lifecycle.

The first step in the lifecycle is the initial inception. The first form the data exists in is its native form, which depends on the tools that are used to create it. It's very likely that most data are created using different tools and due to their inherent difference, can appear in various native forms. The native form cannot typically be used as is, as there is usually a certain amount of processing that needs to be done. CAD models can consist of extremely small details, which are usually not needed when used in a game engine, due to performance issues. As VEs are dependent on smooth performance, simplifying the CAD-data to an appropriate level guarantees a certain level of performance increase.

After the first level of processing, the data is moved to a storage and development environment. As the native data formats are usually specific to their host tools, even after getting them as outputs, they must be translated to be usable in the VR development software. The collection phase can be equated to structuring, as the freshly translated data is assigned to a collection of similar data. This phase is also when version control (VC) needs to be implemented, as many developers can be given access to same data. The implementation depends on the organizational situation. One possibility is to use the same solution for all data, while the other is having separate solutions for binary and text-based data. The difference between the two is the how the VC works on a technical level. Textual files are easy to control, because comparing differences is easy, and the size of the data is relatively small. Binary files are difficult to compare, because computationally comparing differences is cumbersome and the file sizes can get large. Some solutions have been developed aimed at specifically 3D model version control, such as *3D Diff* developed by Doboš and Steed (2012). Therefore, one solution is to use separate VC solutions for binary and non-binary data.

Combination phase is the creation of the three forms of VE components, using the gathered data. The first form is an asset, which is a combination of gathered data forms. For example, an asset can consist of a 3D model, a 2D texture graphic and an animation. The next level of VE components is a scene, which is a collection of assets into an actual environment that can be interacted with in VR. Scenes are environments, but they don't necessarily have to be constructed of the environment data that the project collects; they can be placeholder environments or abstract places that have no correlation in the real world. The final asset form is a build, which is a collection of one or more scenes. Builds are essentially executable applications, that present either the whole aggregate or its part. The asset hierarchy is further illustrated in Figure 22.

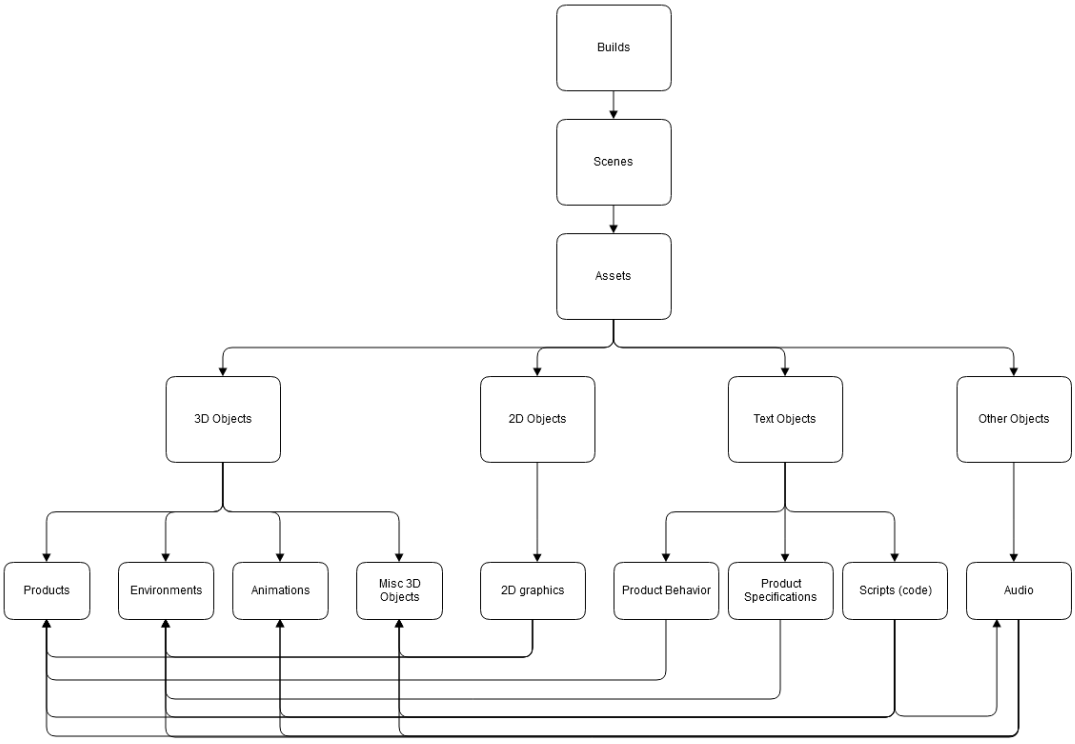

Figure 22. Asset hierarchy diagram.

The asset hierarchy diagram is a deconstruction of all the data forms that the VEs comprise of. The lowest level of the diagram also shows the relationships between the data forms. Essentially, the 3D objects are enhanced by the other data types. Scripts are the most widely used data by other objects, as they provide the functionality. The asset store would also follow this hierarchy, with the data divided into these logical aggregates and they can be sorted and searched based on these qualifications.

Deployment phase means moving the finished builds to an application server and giving access to the users. If issues are detected with any parts of the published build, they can be moved back to the previous phase for refining. Additionally, once new builds are released, the old ones will naturally return to the production step. Eventually some data will become irrelevant, which shifts it to retirement stage. This can happen, for example, by having product data in the system that is no longer needed. The first stop in retirement is moving the data away from any previous storage and into a specific archive. This archive serves as a history of sorts, where the data can still be retrieved if necessary. Finally, when the data is deemed no longer of any use, it is purged.

The vision for the VR app is that access control can be handled on each asset level. If a scene contains an object that only certain users should be allowed to see, this could be adjusted. If certain scenes are only intended for certain users, they could be fitted for that purpose. The POC emulates the access control on a build-by-build basis. Essentially, the client software gives a list of available builds, which include specific scenes. The availability of these builds is dependent on the rights of the user. The system also allows of user groups, so the POC can showcase the possibility of a role-based access control (RBAC). This is further expanded on the in the following chapter.

The purpose of this representation is to provide an overview of what the system requires in its functions and map out the types of acquisitions management level personnel must consider.

5.1.2 Data Models

The first iteration of data modelling considers the immediately important aspects in the POC implementation. These are identified in Table 12.

Table 12. Stored datatypes in the POC.

Data Type	Purpose
Users	Profiles for users of the POC.
Permissions	Access control to VR and CRUD operations.
Groups	Users can be assigned into groups, for logistical and access control purposes.
Builds	The usable VR environments.
Servers	Servers that host builds.
Client (Launcher) software	Used to authenticate and launch VR builds.

User data in the POC environment general information that can be used to authenticate the person and assign them an appropriate level of access to specific functions. Permissions can be interpreted in different ways. On the VR app level, they are used to manage the content that is available to the user. On a web-level, the permissions can be tied to admin tasks, such as is the user allowed to launch new servers and builds or manage users. Groups are for managing users and access rights on a larger scale. As assigning rights to specific actions to every user individually is time-consuming, when the userbase starts growing, having a way to allocate people into roles streamlines this process. The design of the groups can be system- and environment-specific. Typical solutions are role-based access control (RBAC) systems, where upon joining a given system as a user, they are allocated into a role. The role then has a set of permissions. Individuals can also be assigned permissions outside of the pre-set role rights; this is often called responsibility-based access control. For example, a user can belong to a group of developers, while also having some admin-level access.

Once the POC entities were decided, several data models were created. These are further expanded on in a conceptual data model (Figure 23).

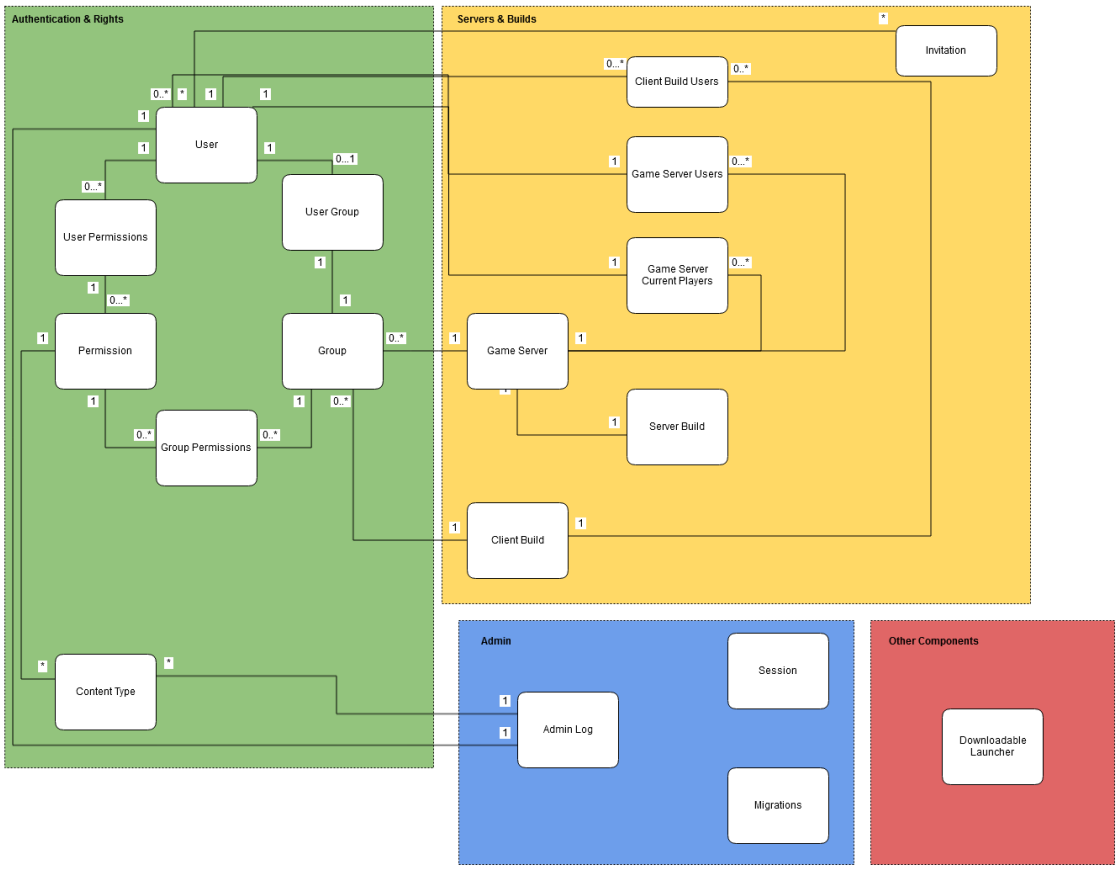

Figure 23. Conceptual Data Model.

The POC model is divided into four categories. Authentication and rights are one bundle, where the user information is stored. Users can be isolated or be assigned to a group. Similarly, permissions can be assigned to individuals and groups separately. This allows for a lot of flexibility on the access control. Content type is related to the CRUD operations (Create, Read, Update, Delete -operations) the users are allowed access to. While most of these are directly related to admin controls, they also affect the available builds and thus the VR data. Admin area has strictly administrative components, which are used for monitoring sessions and getting logged information about actions related to the system. Servers and builds relate to the server environment. Stored areas include servers and clients, builds available on each and users related to each. The reasoning is to have a way to manage, which users can which server and build, and which users are currently inhabiting a server. Invitation component showcases the collaborative aspect of the VR, being a function that allows users to invite others to their scenes. Finally, the downloadable launcher component stores the reference of the launcher file and its current version. Figure 24 shows the logical data model, where the attributes are specified for each data entity.

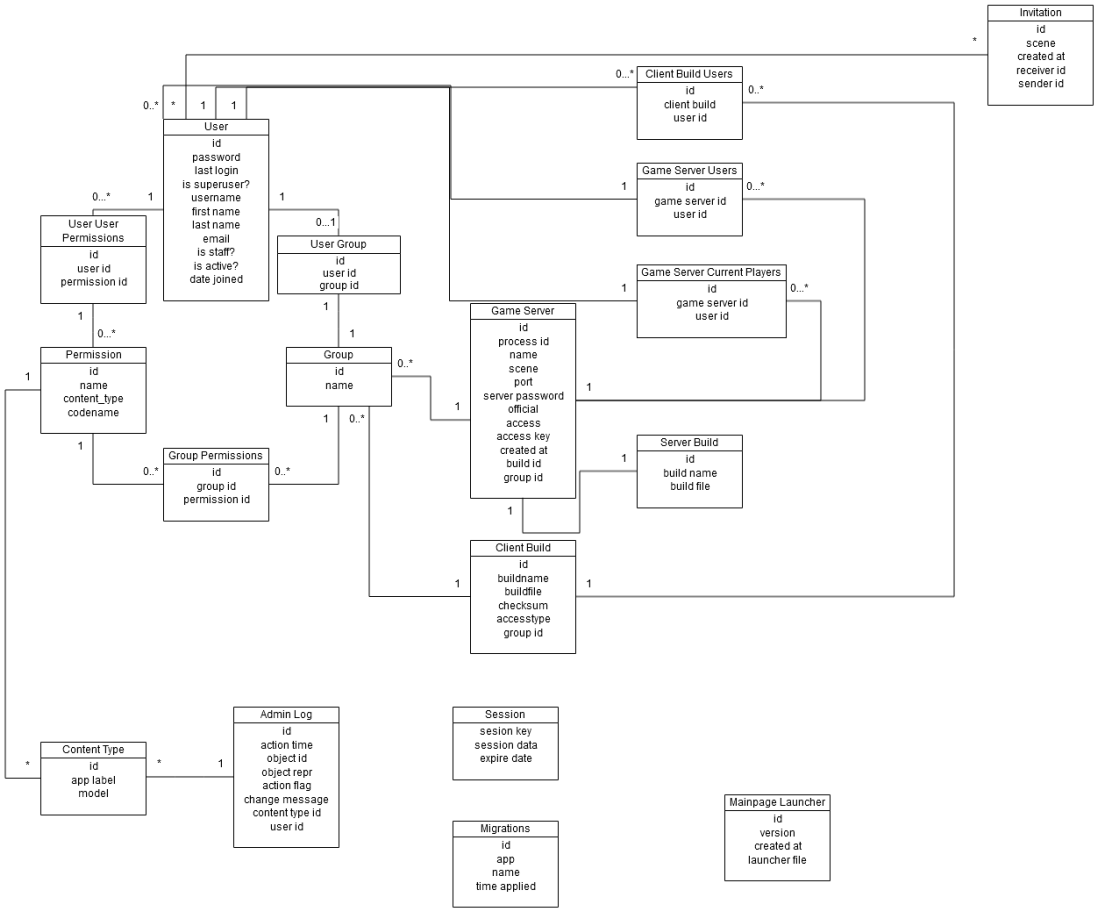

Figure 24. Logical Data Model.

User and Game Server entities are the largest ones in the model. Aside from basic user information, the User entity also stores login data, as well as whether the user is designated as a superuser or a staff member. These are directly related to access control. Staff members are allowed access to admin tools, while superusers automatically have access to all system functions, without needing to have them set explicitly. User ID is used in various other entities, as a means of uniquely identifying the user, for example for the purposes of granting permissions or assigning them to groups. Game Server entity has identifying and technical information about a given server, such as the name, descriptive scene name and which network port it uses. Security can be handled by defining allowed users, setting a password or an access key. Figure whatever3 show the physical data model, which is the final and the most detailed visualization.

5.2 Application Architecture

The application architecture of *Saturn* is consisted of the website and the VR app, which both are responsible for handling the data that the system uses. Website is mostly for administrative functions, such as creating VR servers and designating builds to them. Figure 25 shows the data flow between the applications.

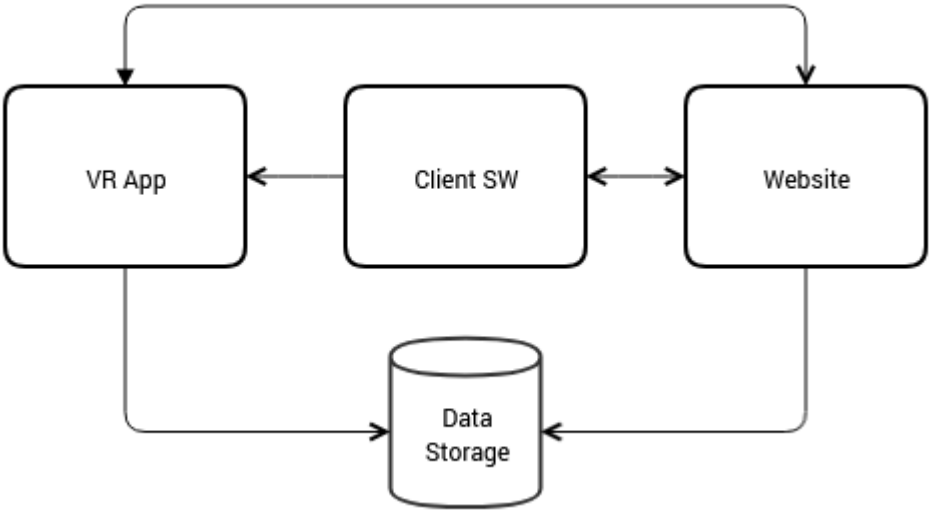

Figure 25. App data flow.

The website is the point of control of the POC system. It allocates permissions to the client software's authentication and by extension, to the available content in the VR application. Website is also connected to a database, which stores data from users, servers and activity logs. Client software is the middleman and the point of access for a generic user. It authenticates the user through the web server, by confirming the user account credentials and afterwards requesting the list of the builds. Once the authentication is completed, the client software connects the user to the VR application. VR application is the main process of the system, using various data to model VEs and allow for rich user interactivity.

The main components of the website are shown in Figure 26.

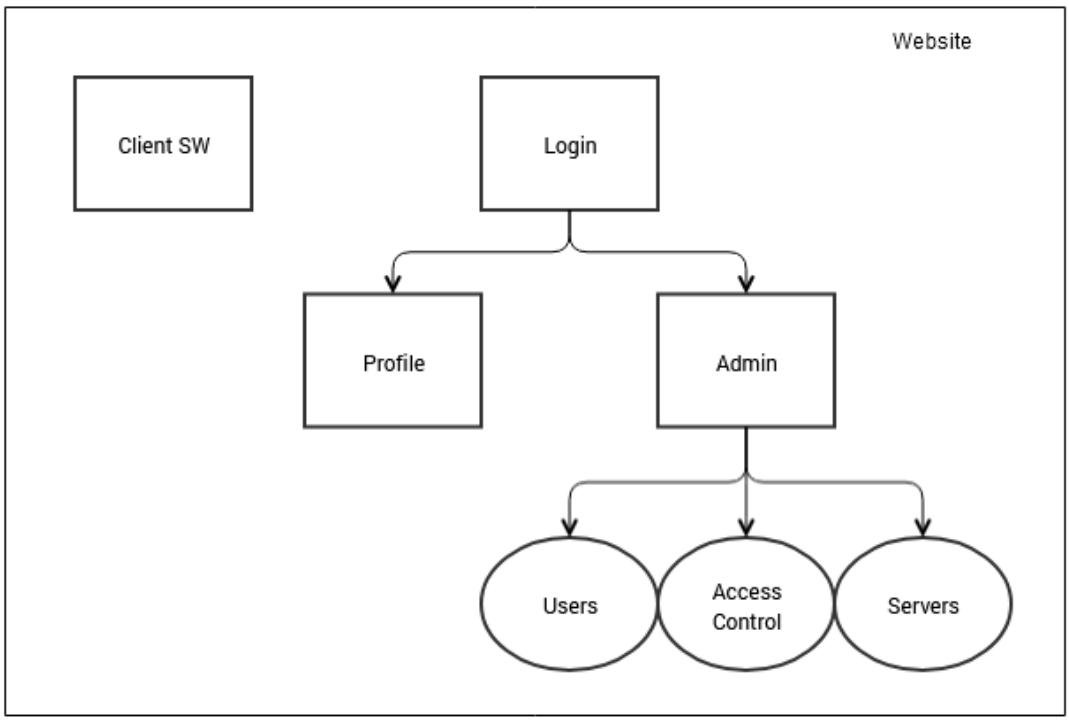

Figure 26. Website components.

While the website isn't planned to have much content to a generic user, it still houses important functionalities in terms of managing the system. On the outermost layer, it has the download-option for the client software, as well as user login. After logging in, a generic user gains access to a profile page, which, in the POC system, only has the option of changing their password. Users with admin access, gain access to a variety of actions. They can manage users, which includes creation and deletion, as well as assigning them to groups. Access control also happens through the admin controls on the website. In the POC, this means that there is an ability to control the available builds on the client software, depending on the user and their assigned level of access. Finally, the website is used for starting and terminating servers for the VR application.

Client software components are shown in Figure 27.

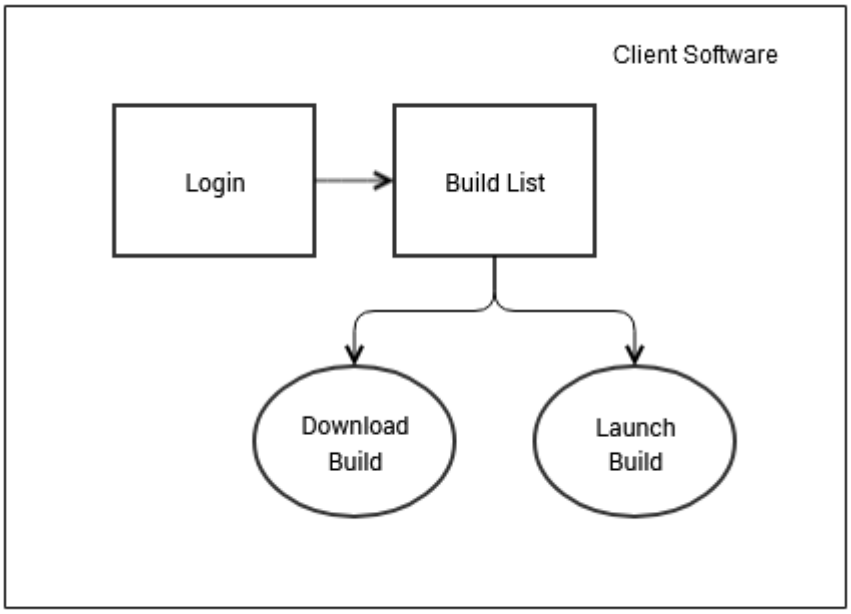

Figure 27. Client software components.

The client software has a very simplistic realization in the POC system, as it's only used as a launcher for the VR application. After logging in, the user is presented with a list of available builds. The user can then download one of their choosing and once the download is done, it can be launched.

The VR application components are shown in Figure 28.

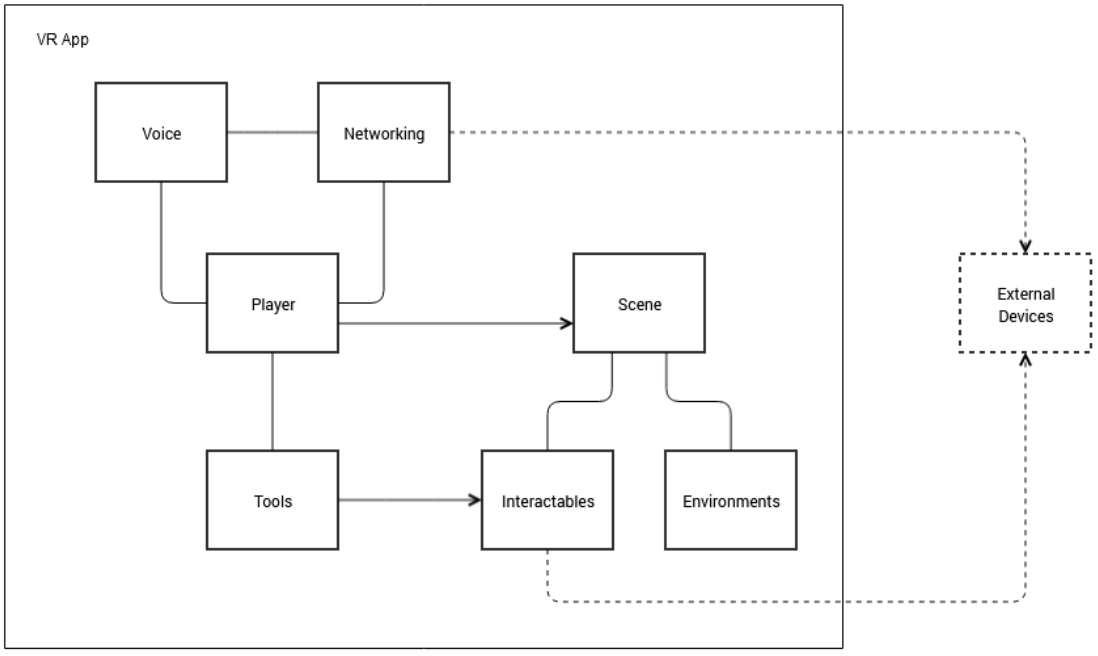

Figure 28. VR app components.

The VR application has several important functional components. The user is the player, who can move around in the 3D environments and interact with objects. To make co-

operation possible, a networking component exist, meaning that the VR servers are operated over a network and the players can appear in them together. Voice component makes use of either external microphones or integrated ones of the HMDs. Voice is the easiest way to communicate and co-operate in the VR environment, so having this possibility is a high-priority. The player can access scenes, which are separate VEs and have interactive objects. The environments can be imaginary or modelled after real-world areas. Interactable objects can be simple buttons, but more importantly, product objects. Player also has some specific tools available, for increased functionality, navigation and immersion. Finally, the dotted line outside of the system to External Devices -component signifies the plan for controlling a device outside of the VR application. Essentially, this would happen over a network, through some interactive measure. This is a tentative aspect of the POC VR application component model; it isn't a high priority at this point but can be attempted if resources allow.

The POC implementation of the app is significantly different to the future vision. The future architecture is a singular app (build), where all the environments are different scenes. The POC system instead is a collection of builds, which are separately delivered to the users in the client software. While the builds already showcase the multi-scene architecture, the multi-build delivery makes the development stage easier. As development is fragmented, and different VR areas are needed in different contexts, using several builds allows for early distribution between developers, who require different areas to be available.

5.3 Technological Architecture

The technological architecture is client-server -based. The first iteration of the POC environment is modelled in Figure 29.

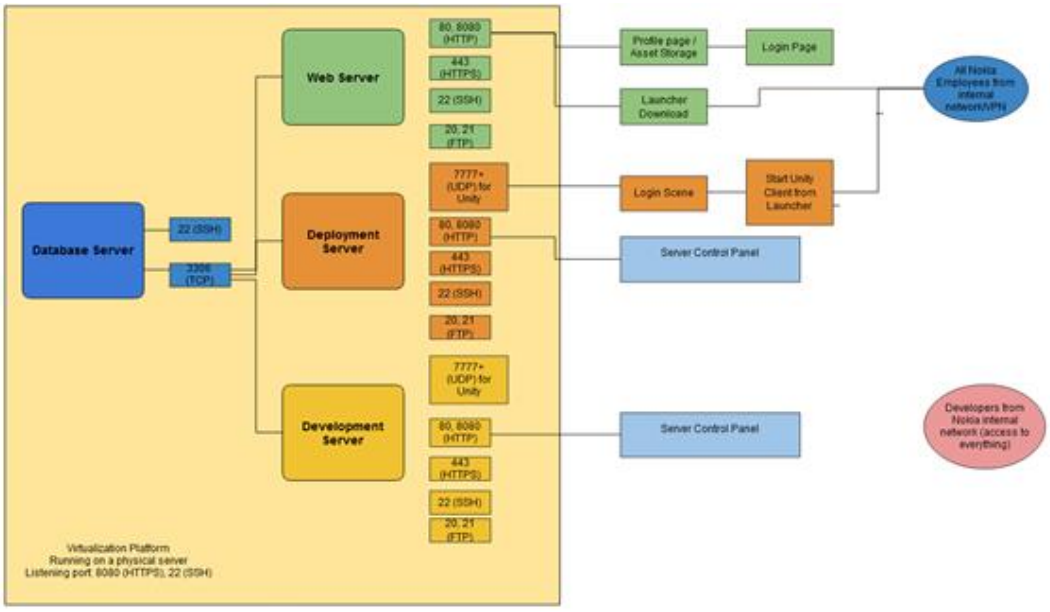

Figure 29. Server environment and networking, first iteration.

The model is essentially tiered into four categories of components. The server environment is the first one. The system is hosted on a physical server, which has a virtualization platform installed. The system consists of four virtual machines, which each host a server. The four servers include database, web, deployment and development. Database is connected to each of the three other servers and hosts all the databases. Web server hosts the website and the tools for user creation, server building

and access control. Deployment server references the servers that the VR builds run on. Finally, deployment server can be used for testing. The benefit of this design is its clear division of responsibilities.

The second category are the communication protocols, which are mostly the same over the servers. Each server has a web-component, which is used via the Hypertext Transfer Protocol, which is a common communication protocol used in Internet traffic. Secure Shell (SSH) and File Transfer Protocol (FTP) are other application layer protocols. SSH can be used for secure communication over an unsecured network and its typical applications are remote command-execution and command-line login. FTP is a protocol for transferring files between a client and a server in a network. SSH and FTP are intended for specific developer use in the *Saturn* system. The VR servers, deployment and testing, use User Data Protocol (UDP) for transmitting data between the user and the server. UDP is comparable to Transmission Control Protocol (TCP), which is common in internet traffic. UDP is generally preferred in real-time applications, such as games, because TCP is much stricter when it comes to loss of data and packets during transmission, which can cause performance issues in real-time applications. TCP is used in *Saturn* between the database server and the other servers because reliability is more important in this case than real-time performance.

The third category are the functions that are carried out over the communication protocols. Most of the traffic is over HTTP(S), as the maintenance happens using web-based means.

The final category were the user groups. The initial model shows two segregations, these being the target organization employees in an internal network and developers. While this is a relatively truthful model in a future implementation of the system, it was deemed an unnecessary division at this point. The POC system was designed to be operated in the target organization facilities. Non-developers inside the target organization were given access to the launcher download and the builds that were considered public. The generic employees would also have login access to their user profile, but no access to asset storage.

However, when the website was created, it was decided that controlling the servers could be done using the website's admin tools, at least for the time being. Being a POC system, this streamlining and centralizing of features creates fewer separate entities that need to be maintained. The second iteration of the environment is modelled in Figure 30.

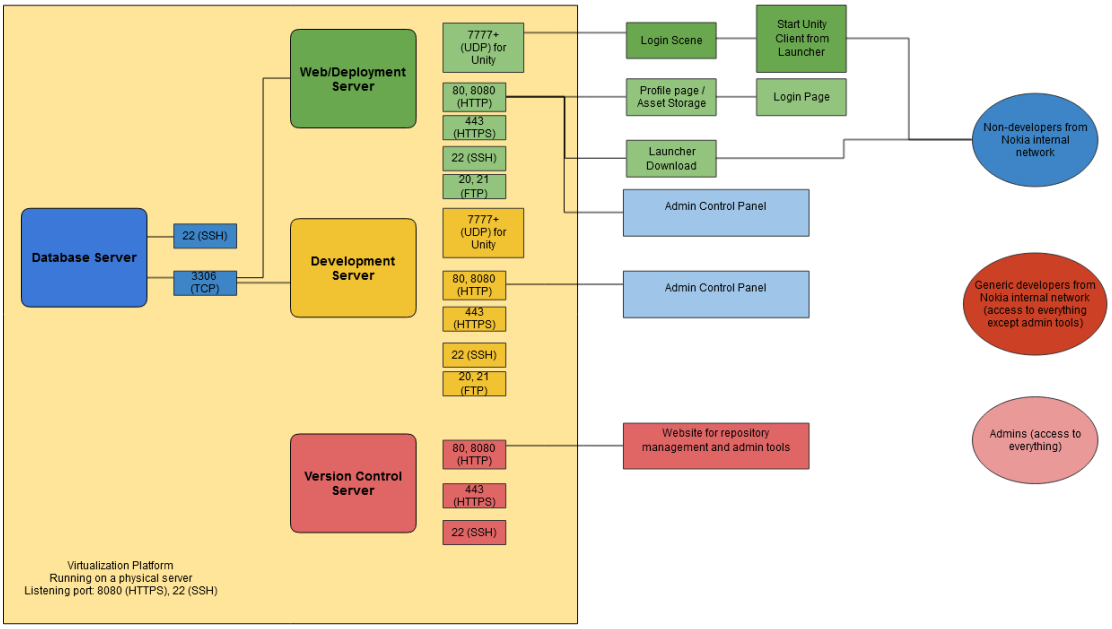

Figure 30. Server environment and networking, second iteration.

The second design combines the web and deployment servers into one. Now the website is the central hub of control, where new users can be created, and their access controlled. Additionally, publishing builds is done using the website. Development server still exist as planned and they too have a web-based control system, equal to the deployment server. Finally, a separate server was added for version control, as it was decided that a self-hosted solution was the only reasonable choice for the project, considering the network limitations that the environment imposed. The VC server was left as a separate component, only being used as a repository for control. The publishing of builds would have to be done manually. The version control server is controlled from its own website.

6. Implementation

The implementation of the architecture design of Chapter 5 can be similarly divided into the three larger architecture areas.

6.1 Technological Architecture Implementation

The POC system was hosted on a physical server, which had a significant amount of processing power and memory to be allocated. As the server architecture showed in Figure 32, the system was divided into three separate virtual machines, which were then allocated computing resources as deemed necessary. The virtual machines were hosted on a Linux-based virtualization platform, ProxMox. The other hardware used during the usage of the system included HTC Vive Pro VR headsets and desktop PCs running Intel i7 processors and Nvidia GTX 1080 graphical processing units. These were selected, because they offer a high-end level of performance for graphically intensive applications, such as VR. Additionally, they would be easy to obtain for other parties that were interested in using the system.

6.2 Data Architecture Implementation

The first phase of the data architecture implementation was to define the data sources in the *Saturn* POC. These are specified in Table 13.

Table 13. *Saturn* Data Sources.

Data	Source
Product models	Internal Designers
Environments	Internal Designers External Developers
Miscellaneous 3D objects	Internal Designers
Animations	Internal Designers
2D visual objects	Internal Designers
Scripts (code)	Internal Developers External Developers
Product behaviour	Internal Labs
Product specifications	Internal Labs
Audio	Internal Designers External Designers

Saturn data is a mix of in-house and external sources. Product models are created by internal mechanical designers. The model design happens using CAD tools, which are used in industrial purposes for creating 3D models. Environments can be hand-modelled by internal designers or outsourced to specialized companies who work with 3D map models. One possibility for creating 3D environments was using point-cloud data (PCD), where an environment is either extensively photographed (photogrammetry) or laser scanned, and the results turned into a coordinate system of points, representing the environment. This is then turned into a 3D model by adding surfaces. The *Saturn* project had no internal expertise in PCD, so external options were explored and used to some extent. For the other 3D objects, animations and 2D objects, the core development team was tasked with creating them. This team also handled the scripting. The product behaviour that was used was acquired from internal labs, that are responsible for testing the products. Audio was used sparingly and created internally.

The VR application was developed using the Unity engine, which is a popular proprietary game engine, used in various kinds of game development. It was chosen for *Saturn*, due to the developers' experience with the environment. As the platform supports certain kinds of data formats, the VR data had to be adapted to fit the decision. Figure 31 shows the adapted data lifecycle from Figure 23 in the context of *Saturn*.

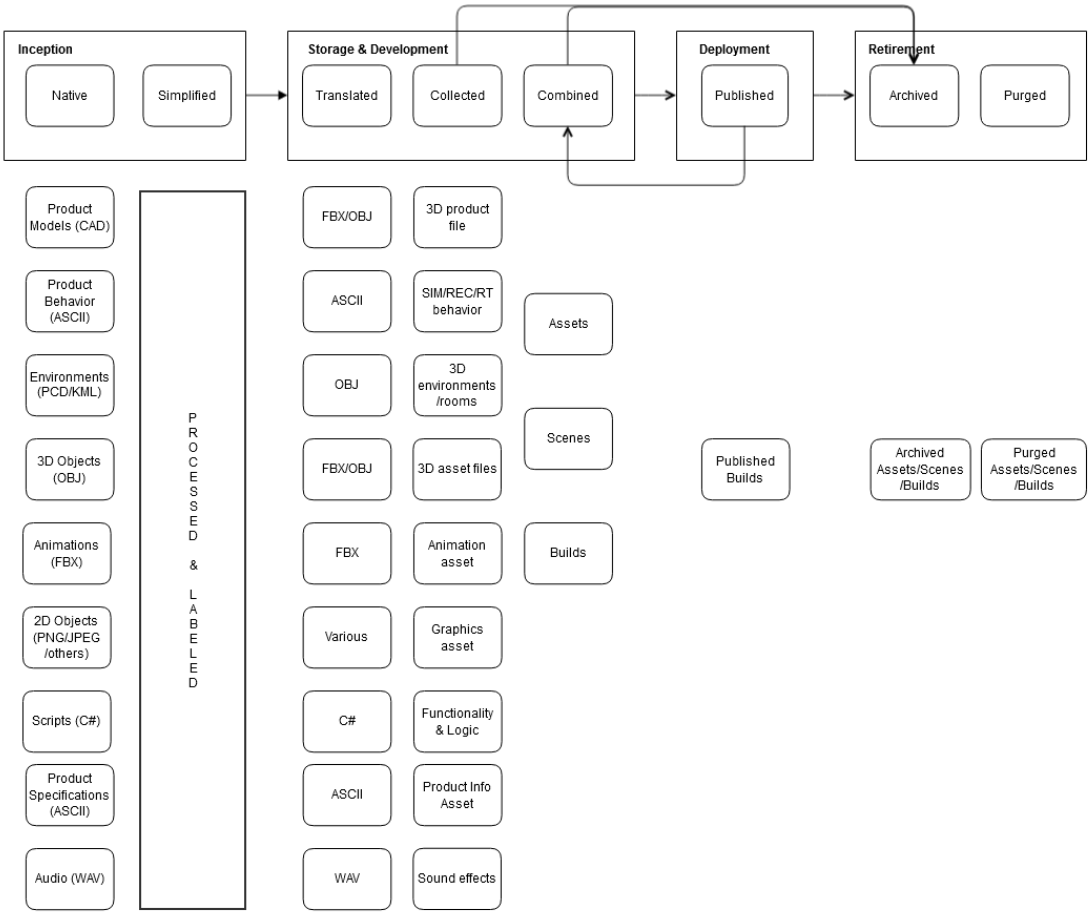

Figure 31. *Saturn* Data Lifecycle.

All the data forms start with their imagined native format. While these formats can vary depending on the tools used, the marked formats were the ones discovered using the tools during the *Saturn* implementation. The simplification phase is data-specific; for the visual data, this means clean-up and reduction, as the native models are typically too detailed to guarantee acceptable performance. Other data, such as ASCII data, can be similarly stripped of anything unnecessary, or restructured in a way that reveals only the necessary information to the developer. Audio data can be compressed as a way of reducing performance penalty.

Collection phase defines the type of asset that the data will become. The term asset is used both in this context, as well as in the combination phase, so it's worth expanding on: In the collection phase, the term asset is used as a singular entity, as a building block that can be used as is in creating the VEs. That said, assets can also be a combination of singular assets, such as a 3D model and a 2D graphic. The asset types in the collection phase of *Saturn* data are self-explanatory. While most of the assets are used in a very easy-to-understand way, such as environments to create VEs or audio to create sound effects, product behaviour is the least understandable asset. The behaviour is collected in the ASCII format, and then implemented using scripts and visualized using graphics. For example, a radio equipment product can have a certain signal power in certain environments. In VR, this can be visualized by adding a graphic to a 3D radio product asset, that dynamically changes depending on the environment. In *Saturn* POC, the decision was to use a singular solution for VC, as the project size wasn't similarly an issue at this point.

As described in the previous chapter, the combination phase divides the singular assets into combined assets, scenes and builds. A scene is a separate area in the VW, such as a town or a lab. Builds exist as both singular scenes and as combinations, the reason being that as an early iteration of the system, having combined scenes isn't always necessary. The final stages of the data lifecycle are executed as the generic architecture design dictates, with publishing and eventual retirement. However, the retirement phase is purely theoretical in the POC system, as no standardized method for this phase was realized in the scope of the research.

The data model of the previous chapter has been adapted in the *Saturn* POC implementation mostly in the same fashion as it was originally designed. The data is stored in a MySQL relational database. The policy in the attributes is unique IDs as primary keys. While having an ID as a primary key is not necessary for every entity, it is used here, because it is the easiest and fastest way to uniquely identify an entity. The main difference in this model is the addition of an "assetstore_asset" -entity. While the asset store wasn't implemented in the POC system, the entity in the physical data model is an example of how assets could be referenced in a traditional relational database. An asset would have descriptive attributes, as well as a file path to its location in a filesystem. The database entity would be used for describing the asset in the asset store.

6.3 Application Architecture Implementation

The website for the *Saturn* system was built using Django framework. Django is a Python-based framework and is both open-source and free to use. These, in addition to the previous experience of the developers, was the reason for choosing it. The core Django framework is an MVC architecture. A Javascript variant Node.js was used for creating the web server, as it was another familiar tool.

The website is designed to be as simplistic as possible. For a basic user, the only functionalities available are the launcher download, a setup guide and the profile page, where the password can be changed. The setup guide gives the user information on what kind of hardware and network requirements they must meet to use the system. Figure 32 shows the admin controls.

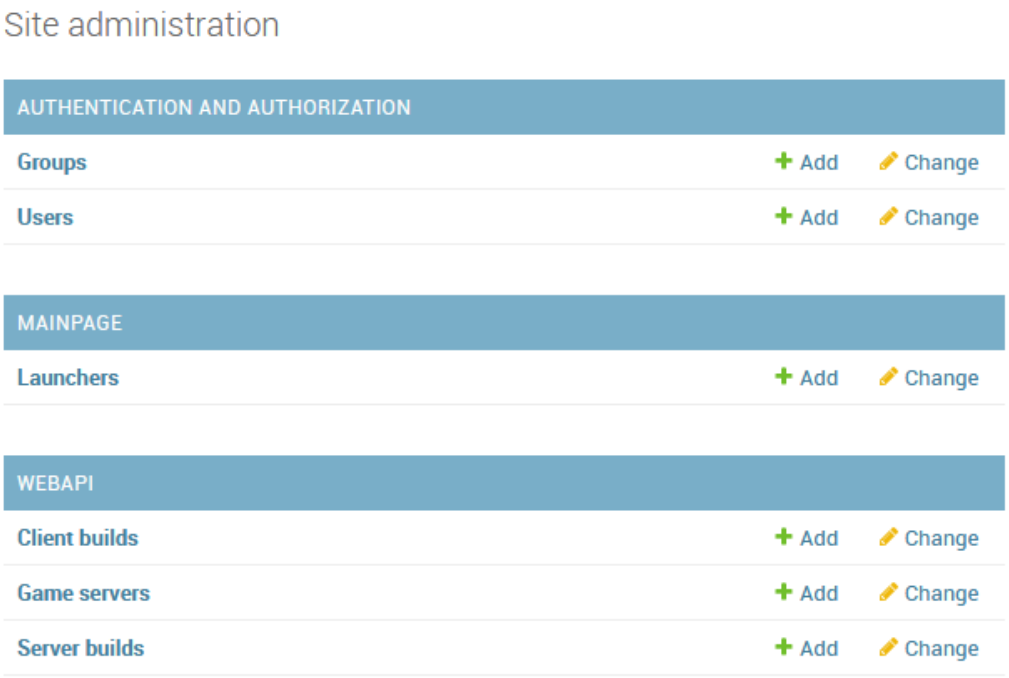

Figure 32. Website admin controls.

The admin controls are divided into three categories, which are authentication & authorization, main page and WebAPI. The first category includes the users and the user groups. The users can be managed as singular entities or assigned to groups in whatever accordance that is deemed fit. While groups can only be managed in terms of access control, users can be assigned some personal and contact information, such as their real name and email address. The user page also logs the registration and recent login dates.

Main page directly correlates to what is available on the frontpage of the site. In the POC system, the only relevant option is the launcher download. In the admin controls, the launcher executable file can be assigned.

WebAPI controls the VR server environment. Client builds determine which VR builds are available on the client software, once the user logs into it. Game servers determine what the users can connect to. Essentially, while the client may show VR builds, only some of them may be running on a server. More builds can be added from the Server builds -section. The server environment is further fragmented from the model shown in Figure whatever, as each scene runs as its own process on a separate server. Therefore, the VR application is less of a singular entity, and more resembling a loose collection of VR scenes. This was due to using Unity as the development environment. To create a virtual world, where each scene persists, and the users can freely either move into separate scenes or join each other in the same one, the scenes must run as individual processes, and therefore as individual servers. The alternative would be having one large environment, where each separate scene would instead be a separate physical location. This, however, would result in much larger environments and further decrease the performance.

Aside from the admin functions, the website is used for downloading the client software. Figure 33 shows the client login page.

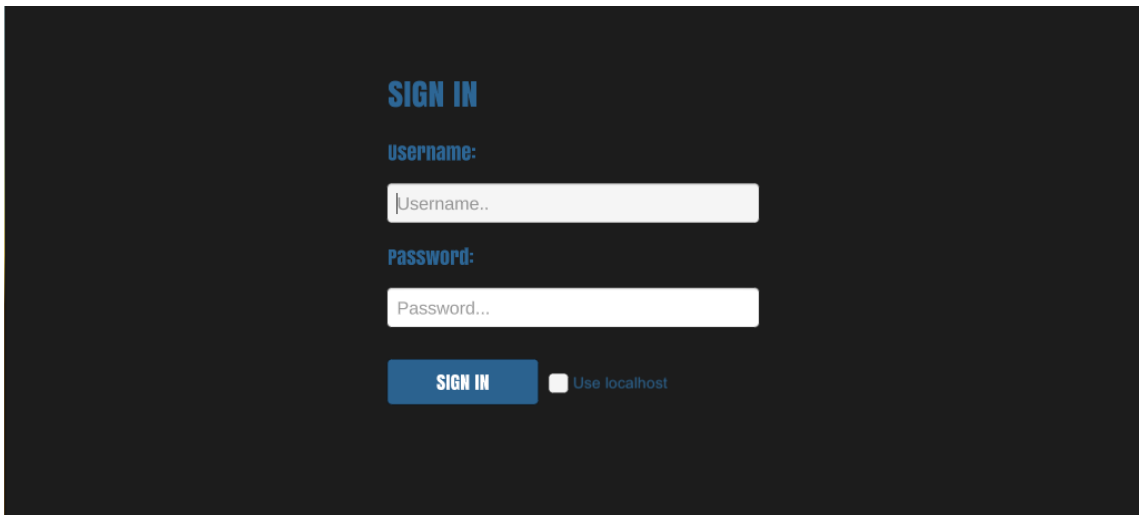

Figure 33. Client software login view.

The landing page of the client software is minimalistic by design and the only function is the authentication. The purpose in the POC environment is to not overload a new user with anything that is directly relevant to accessing the VR app. Once the login process succeeds, the user is presented with the list of available scenes (Figure 34).

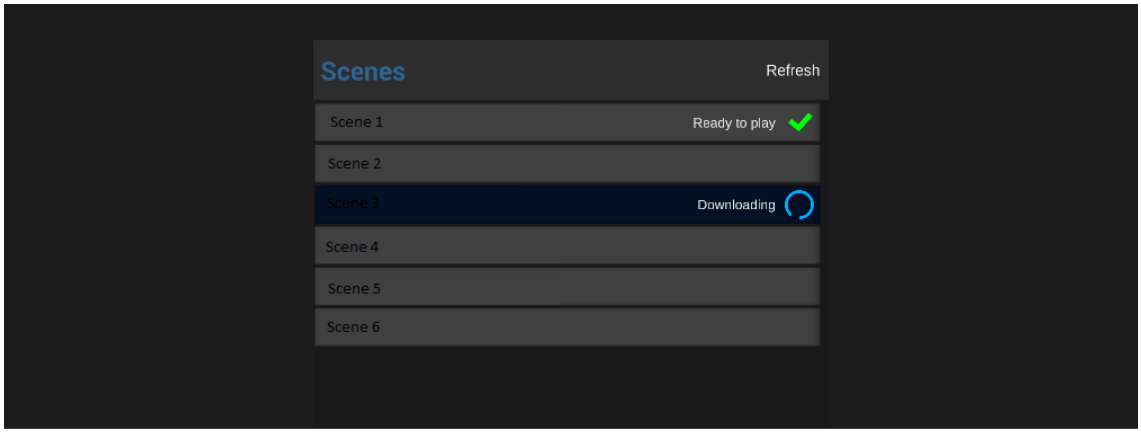

Figure 34. List of scenes in the client software

The scene list that is shown after logging in is depended on what is made available to the user on the website. Clicking on a scene initiates the download process and once the download is finished, the scene is ready to be launched. Launching a scene starts a process of connecting the user to the VR server that the scene/build is running on. If this connection request succeeds, the scene launches. The client process is further modeled in the process diagram in Figure 35.

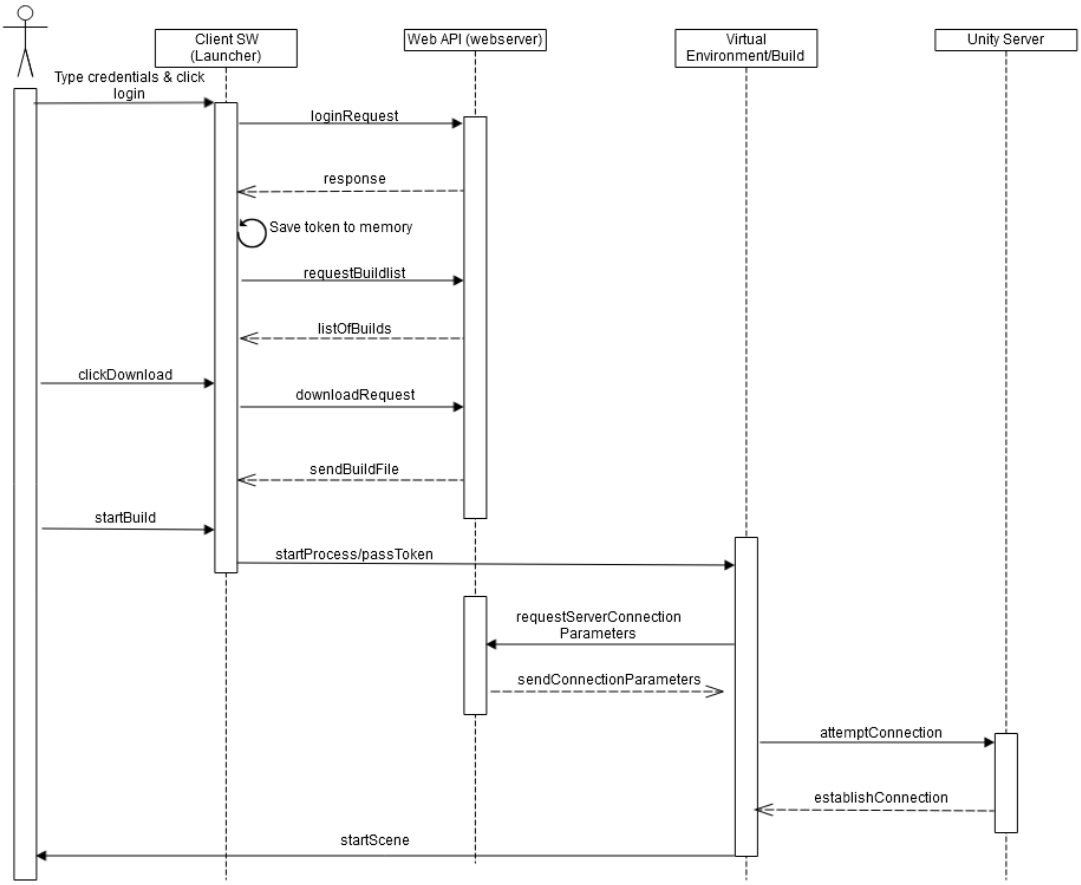

Figure 35. Login procedure in the client software.

The initial communication, after the login information has been inserted, happens between the client software and the web server. The client sends a login request to the web server, which then sends a response. If the credentials are correct, the web server generates and responds with a token, which is saved to client software's memory. In the POC system, the token contains the user ID, username and the token expiration time, and it is used for all further communication with the WebAPI, if there is need for authentication. After the login request, the client requests the list of builds/scenes, which are delivered based on the permissions the user has. Upon clicking a build, the client sends a download request to the web server, which is answered with a file corresponding to the build. Clicking on the downloaded build starts the process of connecting to it. When starting the build, the token is also passed along as a start parameter. The build then communicates with the web server, asking for server connection parameters. If the build is running on an active server, the web server sends the connection parameters to the client as a response. The client then connects to the Unity server, that the VR build is running on, using the received connection parameters. The final stage is validating the connection to the Unity server by decoding the token. The token removes the need for the Unity server to communicate with the WebAPI in this phase, as both know the secret key for the token, which is used for decoding it. If the token validation succeeds, the user is spawned into the VR app as a player object. If the token validation fails, the connection to the server is terminated. Finally, the VR build is started, and the user can start interacting in it.

The VR application has several of the planned functionalities. The GUI is used for selecting an environment that the user wants to visit. The POC system has mostly conceptual environments. The GUI can be operated using the VR controller.

While the environments are mostly conceptual, they still meet the requirement of having interactive elements and having possibility for co-operation, using tools and products. Figure 36 shows an example of a mobile phone tool.

Figure 36. Mobile phone tool.

The mobile phone tool can be used for taking pictures and sending them to other users' mobile phone tools. In the future, this can be used as a quick way to store and share points of interest in VR, such as product designs or plans. As co-operation is the goal in the system, it is further enabled by other means (Figure 37).

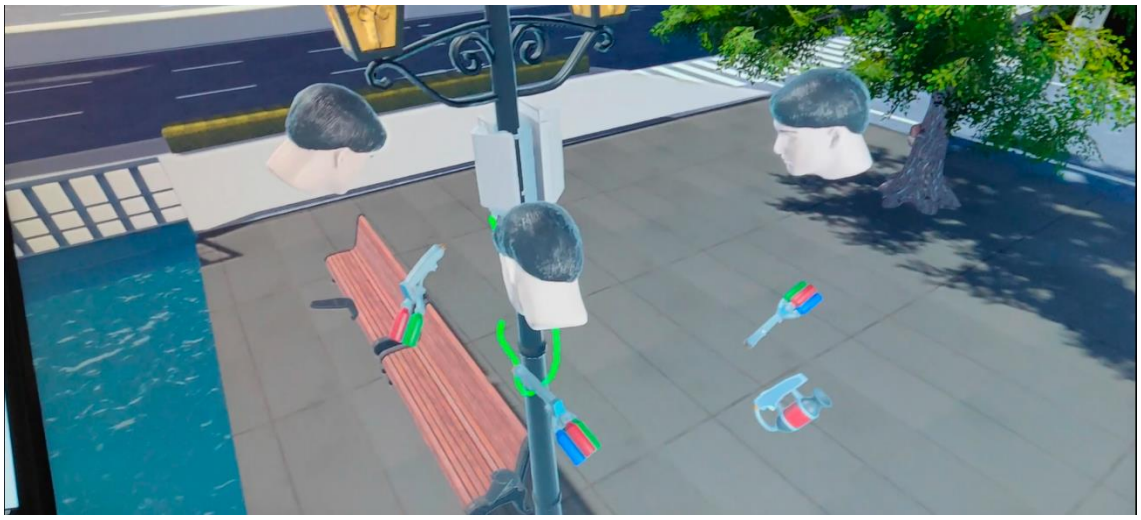

Figure 37. Co-operation in the VR application.

As the figure shows, multiple people can work co-operatively in the same VE. Pictured are two types of tools: a drawing tool and a movement tool. The drawing tool can be used to draw simple green lines and used for whatever purpose needed. In the figure, the green lines are used to indicate possible cable solutions on a radio product attached to a street lamp. The movement tool is used for travelling larger distances, as opposed to the very short distances that can be traversed by walking in the physical room. The figure also acts as proof for global availability, as the three users were connected from different locations.

7. Findings

The architecture design was used in a POC implementation of *Saturn* IS at the target organization. The implementation process was handled by a small team working at the company's Oulu facilities, of which the author of this thesis was a part of. The author was responsible for developing and documenting the architecture design and offering feedback and testing during the implementation phase. The evaluation of the system was done as a case study, using the system through its development with other developers and during presentations. The author took part in the evaluation by using the system, presenting it during exhibitions and by taking feedback from users.

The VR asset creation process, and the subsequent data lifecycle model, was the least developed aspect of the architecture in the implementation-phase, and therefore the most difficult to properly evaluate. While the model is in-line with the knowledge base in its design, adapting it into a working system with appropriate level of automation was not possible due to limited resources. The model did succeed in helping the project toward developing an understanding of what kind of data is needed for VR material creation, what kind of internal and external sources create it and what kind of forms the data goes through before it becomes usable. These steps were already implemented manually, getting data from the identified sources and turning it into usable forms, but no explicit system was created for data operations. The asset storage idea, which would have been a centralized system for VR data sharing, could not be implemented, due to a lack of resources.

The physical data model was deemed sufficient, despite being very rudimentary. The relationship between the users, permissions and the VR content mean that the POC could be used to showcase the concept of access control and the ability to separate people into distinct groups. The access control worked on a build-to-build basis, but the developed solution results in a lot of manual work.

The final technological architecture had some advantages over the earlier iteration. By combining the website with the VR server control, the whole POC system could be managed from one central location, which decreased the amount of separate entities that would have to be managed. Considering the POC was developed with a small group, this logistical change was a benefit. The virtual machine (VM) model made managing the separate systems relatively easy. Using a proprietary physical server as the platform also increased the amount of control the development team had over the environment, as all data resided in the physical space that the team had control over. While the solution doesn't scale in a way that is easily sustainable, it worked as a baseline implementation. The scalability is an issue that affects most of the other VMs as well, namely the VC machine, since the designed solution is also centralized, meaning that assets of all kinds were stored there. As binary assets can quickly grow larger in terms required storage space, the POC solution may suffer as the system develops. The current remedy is to reallocate space for the VM, and then look to physically add more storage space in the form of hard drive expansions or additional physical servers.

The application architecture supported majority of the intended functions. The second iteration of the technological architecture and the combination of the website and the VR servers, made the implemented system more centralized than originally intended. Functionally, the architecture fulfilled the purpose of the applications, with the major

issues appearing in terms of optimization. None of the architectures considered security explicitly, meaning that in the implementation phase, the security solutions were left to the judgment of the developers. In the POC system, security issues were mostly neglected, as resources were limited and the small-scale internal distribution of the POC system made it not an immediate priority. Following the server environment model, the first steps toward a more secure system would have included shifting the HTTP connections to the secure HTTPS connections. In this case the concept of “security through obscurity” was adhered to.

While the architecture didn’t specify the development tools, using Unity morphed the VR application into a collection of scenes, each of which were running in their individual servers. This made the access control easier, as there was no need to create AC methods into the VR application. The downside is that each scene needs to be separately assigned to a server and then activated, increasing the amount of work. The project had several situations where the server environment was littered with duplicate builds and scenes, indicating that management could become a concern.

The architecture and the implementation can also be analysed using the requirements engineering, and the list shown earlier in Table 9. The POC system and how it met the requirements are shown in Table 14.

Table 14. Requirements revisited.

ID	Requirement	
FR1	Website allows for user login	
FR2	Website has a downloadable client software	
FR3	Website has some level of technical support	
FR4	Website has some implementation of asset storage	
FR5	Client software can be logged into	
FR6	Client software can be used to launch the VR app	
FR7	VR app has some locations the user can visit and move around in	
FR8	VR app has some products that can be interacted with	
FR9	VR app allows for co-operation	
FR10	VR app can be updated	
FR11	New builds of the VR app can be deployed	
FR12	Maintenance tasks can be done for the website and the servers	
QR1	Multiple users must be able to interact in the same environment	
QR2	Response time of below 200 milliseconds	
QR3	Framerate of 90 frames per second	
QR4	Downtime to be minimized when servers are in maintenance	
QR5	Automatic error messages and reports	
QR7	Always online	
QR8	Encrypted connections	
QR9	Encrypted databases	
QR10	Access control in a general sense	
QR11	Servers updatable	
QR12	Additional servers for backup	
QR13	Designs in the VR aimed toward helping usability	
Implemented	Not Implemented	Partly Implemented

Out of the 12 functional requirements, only two were left either partially implemented or incomplete. The updatability of the VR app was envisioned to mean essentially patches of content, where the baseline app was supplemented with changes as they were made available. Instead, the POC app essentially works as a collection of builds, instead of a singular application. This means that if changes are made, they will affect entire builds and requiring them to be made available again after the updates. This inherently means that the operating of the application requires a lot of manual work and possibly constant reuploads of the same builds, even after small changes. The functional requirement that was completely unimplemented was the asset storage, which was supposed to be a hub for developers and content creators to store and share their work. In the end, the lack of resources meant it could not be implemented.

The quality requirements had more variability in terms of reaching the intended target. Out of the 13 requirements, five were met, four were met partially and four were not implemented. The VR application met the requirement of allowing more than one user co-operate simultaneously, in the same environment. The VR app also met the intended response time limit of below 200 milliseconds, which was tested with several setups of hardware, as well as in several geolocations. In terms of maintainability, the access control system meets the “general” level, since it allows precise control of which user can access any available VR build. Finally, the servers themselves could be updated, as they were designed to be virtual, which can be created and modified easily.

The partially implemented requirements concern the availability and security. While the system was technically available always, by having the virtual machines hosting the server running constantly, there were no specific steps taken to ensure constant availability. In the case of failure, there were no measures of automatic recovery. This is the same for server redundancy, as the POC was hosted on a single physical server. The partiality of the implementation in this case means that while there was a backup server available, this server wasn’t readily available as a backup, and instead it was designated for other purposes. The security requirements were left mostly neglected. While the database did have some preliminary encrypting of passwords, no security measures were specifically developed to protect the system from unauthorized use or access. Connections were designed to be encrypted in the architecture design, but in the implementation, these were left unimplemented, due to needing to allocate resources elsewhere.

The unimplemented requirements affected performance, maintainability and usability. The frames-per-seconds target of 90 was not met definitively, and due to the unoptimized nature of the VR app, this performance metric was uneven and produced different results depending on the build. While the performance was passable for all the use cases, including meetings and presentations, further work to ensure reliably smooth framerate would alleviate the issues with cybersickness. Maintainability related requirement of minimized downtime during server maintenance was not a realistic goal in the end, as the resources did not allow for developing a system that would guarantee swift recovery from any possible maintenance. Finally, while usability was thought to be a critical requirement, due to the general unfamiliarity of VR applications, the focus had to be shifted toward function over usability. Therefore, while the VR environments were built to only accommodate their intended purpose, therefore lessening the possible distractions, no development was done using any available usability heuristics or methods, nor were the intended error messages created. These were not a critical issue, since the development team worked in close co-operation with any new users, being able to provide aid, instead of building any into the VR app itself. Therefore, neglecting usability can be supplemented by providing adequate hands-on support, although this approach requires manual work.

8. Discussion

The research problem of the study was to find out, what kind of architecture supports a digital information system, where virtual environments are utilized. As this is a large topic, this was examined using three sub-problems, focusing on finding a suitable data architecture, technological architecture and an application architecture. The data architecture presented in this research divides itself into VR material creation process, as well as into traditional data models, where authentication, authorization and server data are handled, making it the backbone of the access control. The technological architecture is a single-location system, where virtual machines are used to create a collection of servers, with a division of responsibilities. Finally, the application architecture divides the POC into a co-operation of three applications, these being the website, client software, and the VR application. The website is the core, which is used for registering users and their access rights, launching VR servers and making builds available, as well as distributing the client software and technical support for new users. The client software is the authentication portal into the VR application, while the VR application is the functional center of system, enabling the VR experience. VR application has possibilities for users to co-operate over a network, using voice communication, and use various tools and products of the target organization in several environments.

The data architecture is difficult to define explicitly (Tupper, 2011), but generally, it consists of the ways data is used, stored and managed. This research divides the data architecture into two areas, where the VR material creation as well as traditional data storage in databases are considered. The VR material creation process uses applied models of data lifecycle and is visualized using a four-tier model, where data goes from inception and processing, to publishing and retirement. As such studies on VR-based systems are sparse, it is difficult to find comparisons in terms such data models. While the data that the VEs use are categorized in this project's scope, it still considers the possible data on a large scope. This model is supported by theoretical frameworks for data management, but as the implementation in the POC system is mostly theoretical, it is difficult to evaluate. Currently the lifecycle model can be applied manually, but automation is a must for a larger system. As Stark (2016) writes, automation in data management decreases much of the chaos that is present otherwise.

The POC technological architecture consists of a physical server, a virtualization platform and three virtual machines, which operate the servers for the virtual worlds, the website, the development purposes and the version control. As Cheng (2014) writes, virtualization of servers is a common strategy in current IT operations, as it creates a possibility for an improved allocation of resources. Instead of using several physical servers, virtualization allows the use of just one, in which several VMs operate as if they were isolated computers. Virtualization platforms also have several useful features, that help managing the availability of VMs, such as migrating running virtual machines to different hardware, template creation for subsequent VMs, flexible storage and networking options, and high availability and backup tools. While the POC system utilized on-premises server hosting, cloudification is another form that is increasingly popular. Whereas on-premise solutions require for the service provider to manually prepare and maintain a host of physical servers, a cloud-based solution is handled by an external cloud service provider and the resources allocated to the service as needed. Kaufman (2009) notes how both service-oriented software, as well as virtualized

software and hardware is utilized in the cloud. For a system such as the POC designed in this research, cloud environment would offer several ways to address its current problems. For example, as the POC does not scale in its current state and its security details are poorly executed, a cloud-based solution would offer remedies very easily. However, Pahl, Xiong and Walshe (2013) note on the possible challenges of the migration from on-premise solutions to the cloud. These include technical- and business-level issues, such as need for re-architecting if whole applications are migrated, as well as having to explore the intricacies of integration, security and monetary flow.

The application architecture of the POC system utilizes a model seen in online games. The website is the knowledge center, where the user can learn the basics and download the client software, which is used for accessing the application. While *Saturn* is not a typical business application, this separation of responsibilities was intuitive and easy to understand for first-time users. The assembly of these components is largely dependent on which tools are used for creating them, as any chosen alternative brings their own possibilities and limitations. In the POC of *Saturn*, the website could be made into the central authority, due to the chosen development framework, while Unity engine necessitated the final server implementation due to its attributes. Therefore, proper research should be put into various tools and platforms that are available for creating a system like *Saturn*, as the decision may impact what functional decisions can be realized.

From a theoretical standpoint, *Saturn* manifests attributes of virtual enterprises, which are a collection of organizations that come together to combine their competencies for a collective gain. Cardoso and Oliveira (2004) note how the concept of virtual enterprises has been used to describe common business-to-business operations, such as outsourcing and supply chains, which could be a long-term goal for *Saturn*, since the VR material can be created by various specified entities and co-operationally implemented in the virtual environments. As Camarinha-Matos and Afsarmanesh (2001) write, virtual enterprises also act without being a legal entity or having a physical base of operations.

9. Recommendations for the target organization

In this chapter, specific issues and future directions for the PoC are discussed.

9.1 Future Possibilities

While there is inherently a lot of work to be done on the POC system, the research done shows some more or less obvious points of improvement and possibilities (Table 15).

Table 15. Future possibilities for POC system.

Area of POC	Possible future direction
Access Control	Move to Target organization ID
Server Environment	Cloud-hosted Microservices vs Monolithic
Hardware	Virtualization
Data operations	Big Data collection

As it stands, the POC system's way of access control is manual and work-intensive in the long run, as the implementation includes no automation in the creation of new user accounts. All target organization employees have a personal organization ID, which is used for authenticating into various already existing company tools. Therefore, a natural way of extending *Saturn* would be to modify the user profiles to work through organization ID.

The server environment of the POC is rudimentary and has several issues, when it comes to a long-term sustainable system, as it has barely any redundancy and it's not designed to scale. As cloud-based services are becoming more and more the norm, this would be natural avenue for *Saturn* as well. The target organization is already using Microsoft's Azure platform in some operations, meaning that there should be applicable support available for any cloud-based endeavours. Using these resources, finding a feasible way to move from a physical server into the cloud would alleviate many of the issues that the current system has, such as scalability and security.

Aside from the host hardware, the system design could benefit from reworking, for example, using microservices as an overarching design principle. Figure 38 shows a preliminary example of a microservice-based architecture.

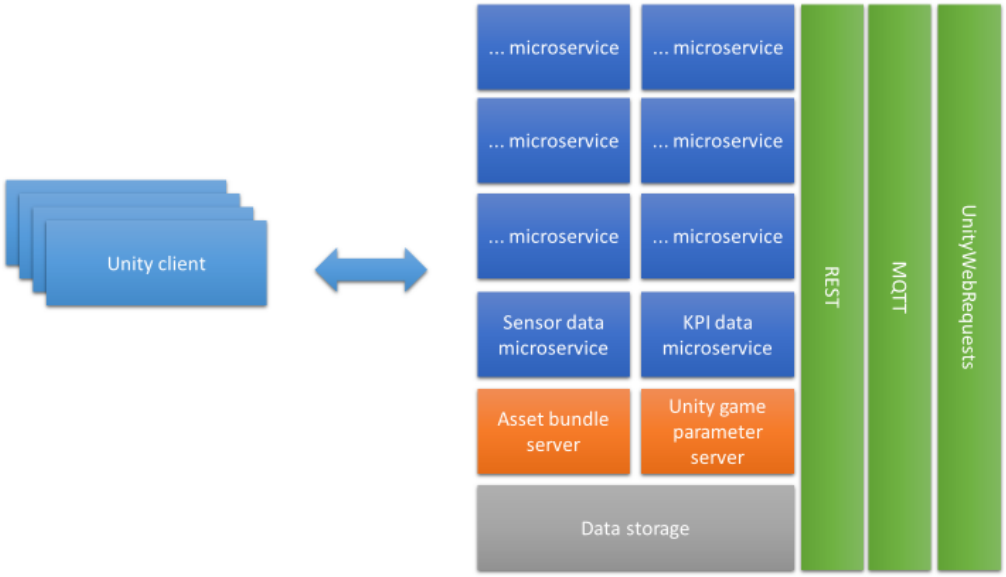

Figure 38. A possible microservice-based implementation (Esa-Matti Sarjanoja).

Essentially, the idea behind microservices is dividing the applications into smaller parts, each of which can be developed separately and essentially exist in isolation from others, while still delivering its function to benefit the system. The VR application, for example, could use microservices by having each larger functionality running as its own microservice. Additions in the Figure 43 include separate microservices for sensor data and key performance indicator (KPI) data, as well as the server instantiations for the Unity VR servers. Microservices could be logically divided into virtual machines or containers, and then monitored and maintained. Microservices-model are useful in the world of continuous integration and development, when rapid changes are constantly being made to an evolving system.

Another interesting future possibility is hardware virtualization. Currently, expanding the use of the POC comes with the added cost of needing to buy hardware capable enough to comfortably run the VR application. This is costly both in terms of money and logistics, as the hardware takes up a significant amount of room, especially when more users are wanted in the same physical location. Hardware virtualization would essentially mean creating machines, with several instances of an OS, such as Microsoft Windows, each instance with enough performance to run the VR application. This would require creating experimental PCs, with multi-core CPUs, with several powerful GPUs, where each GPU is designated to a single user. Both Nvidia (Feltham, 2017) and Intel (Ung, 2017) have produced POC systems such as these on their own. While these would save both physical space and money in the long-term, they would come with a high short-term cost, as they would be custom-built and tested, as there is no set way of building such as machine.

System such as *Saturn* has the potential to generate a lot of so-called Big Data, such as time spent in the application, interactions between users, quitting points, various server metrics and so on, which then can be used for creating a better product in the long-term. Big Data comes with another question between self- and cloud-hosted implementation. In this case, cloud-hosting has a lot of logistical pros over a self-hosted solution. While self-hosted Big Data operation can be cheaper long term and allows for more fine-tuned control of the environment, there is a higher level of expertise and work needed in the setup and maintenance, while security and failsafe are issues that are harder to mitigate. Cloud-based solutions are faster to setup and there is less to worry about in terms of

availability. Scalability, as with the *Saturn* system as a whole, is both cheaper and easier to manifest in a cloud environment as well.

9.2 Unknown Areas

During the research, several areas of intrigue were come across, but were left in a state of relative uncertainty. Still, they may offer future potential, if the solutions that they require become more refined and accessible. These are briefly outlined in Table 16.

Table 16. The unknown areas.

The Unknown	Description
Cloud Streaming	Run the VR app in the cloud.
External interfaces	Connect *Saturn* to outside machines and software
Machine Learning	Enhance *Saturn* functionalities
Artificial Intelligence	Enhance functionalities and user experience

Cloud streaming was brought up to remove high hardware requirement that the VR application has for its use. Basically, instead of running the app on the user's machine, it would instead run in the cloud, using adequate hardware. Therefore, the user could connect to the app using essentially whatever PC and run the app comfortably. While this has been tested and is even used as a service for non-VR games, its VR applicability is uncertain, and the solutions are unrefined. While it is in a way in a similar situation to HW virtualization, it is yet unclear how feasible cloud streamed VR gaming is on a larger scale, whereas the virtualized HW is a more mature concept. It would most certainly come with a high initial buy-in, as creating such as system would be very experimental, unless new breakthroughs happen in the area.

Machine learning, and AI are uncertain areas in terms of what kind of implementation should be considered in *Saturn* and how they would provide value in the long term. Neither concept was explored much at all, so they are currently the definition of an unknown possibility.

9.3 Must-haves in the short term

The previous two sub-chapters explore some feasible and currently uncertain areas of future improvement, but there are also some areas of the POC that are necessary to sort out, before any kind of larger-scale deployment is initiated. These are outlined in Table 17.

Table 17. Must-have requirements.

Requirement	Description
Security	Encryption Fail safes
Redundancy	Server environment rework
Scaling	Server environment rework
Automation	Data operations

The current security implementations of the POC system need to be assessed carefully. At the minimum, this would entail encrypting every critical information, such as databases and connections.

The current server implementation doesn't allow for easy scaling or have much mitigation against failure, aside from having VR builds backed up or backing up entire virtual machines. For a larger deployment, it is mandatory to have a working solution for both, to accommodate larger userbase and server load. Cloud-hosted environment, as mentioned, offers both benefits. Finally, the data operations of the POC are manual and tedious; while there is an understanding of what data is needed and where it comes from, there are no ways of easily get the data from one entity to another. Essentially, different data sources, such as the internal ones at the target organization, as well as the external ones, should have some means of easily distributing the data they create into the developers of the IS itself. Friston, Fan, Doboš, Scully and Steed (2017) developed a method for dynamically loading 3D assets in Unity at runtime, over the Internet. Their solution was aimed at alleviating the issue of industry use of VR, where teams in various time zones and geographical locations would need to work together in an iterative development process. Scully, Doboš, Sturm and Jung (2015) developed a web-based 3D asset repository, meant to manage digital 3D models, which is like the asset storage idea presented in this thesis, although the asset storage wasn't directly meant for 3D assets.

10. Conclusions

The research's topic was explored using the following research question: **What kind of architecture supports a digital information system leveraging virtual environments?** As this is a large and expansive topic, it was divided into three sub-questions, targeting data-, technical- and application architectures. The data architecture that utilizes access control and simple databases for critical data are sufficient for a working POC system, despite being rudimentary. While this research couldn't support the designed data lifecycle model with a physical implementation, the knowledge base supports its focus on determining the source of data, how it is used and how it manipulated to meet the goals of the organization. Technical architecture using virtualization and VMs as the server base is an effective method of creating a simple multi-server architecture, where each component can exist in isolation, while still working to support one-another. Finally, the application architecture of a website, client software and the VR application offer a simple, but effective trinity of responsibilities. Website acts as the centralized knowledge-hub, the client software as the authorization portal and the VR application as the content delivery platform.

The research offers one solution for creating a POC system of a VR-based IS. It can be used as is as a blueprint and as a springboard for further development. Due to organizational limitations, the research is limited in its intended implementation, as some of the planned features couldn't be empirically tested. These include the asset storage and the overall data lifecycle automation.

Future research possibilities include exploring similar IS in a cloud environment. By using cloud, the solutions could be immediately scaled up to accommodate more features. Hardware virtualization already happens with servers, but the same could be further explored with user-HW, lessening the end-user barrier for VR-application usage. Cloud-streaming of VR content is a similar research avenue. Additionally, each of the three sub-research questions of this research can be further research on their own.

References

Adams, K. M. (2015). *Nonfunctional Requirements in Systems Analysis and Design* (Vol. 28). Cham, Switzerland: Springer.

Ameri, F., Dutta, D. (2005). Product lifecycle management: closing the knowledge loops. *Computer-Aided Design and Applications*, *2*(5), 577-590.

Ball, A. (2012). Review of data management lifecycle models.

Bell, M. W. (2008). Toward a definition of 'virtual worlds'. *Journal For Virtual Worlds Research, 1*(1).

Camarinha-Matos, L. M., & Afsarmanesh, H. (2001, July). Virtual enterprise modeling and support infrastructures: applying multi-agent system approaches. In *ECCAI Advanced Course on Artificial Intelligence* (pp. 335-364). Springer, Berlin, Heidelberg.

Cardoso, H. L., & Oliveira, E. (2004, October). Virtual enterprise normative framework within electronic institutions. In *International Workshop on Engineering Societies in the Agents World* (pp. 14-32). Springer, Berlin, Heidelberg.

Cheng, S. M. (2014). *Proxmox High Availability.* Packt Publishing Ltd.

Chung, L., Nixon, B. A., Yu, E., & Mylopoulos, J. (2012). *Non-functional requirements in software engineering* (Vol. 5). Springer Science & Business Media.

Cruz-Neira, C., Sandin, D. J., DeFanti, T. A., Kenyon, R. V., & Hart, J. C. (1992). The CAVE: audio visual experience automatic virtual environment. *Communications of the ACM, 35*(6), 64-73.

Earnshaw, R. A. (Ed.). (2014). *Virtual reality systems*. Academic press.

Harsh, O. K. (2008). Reusable data, information, knowledge and management techniques. *Journal of Knowledge Management Practice*, *9*(3), 15-24.

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2008). Design science in information systems research. *Management Information Systems Quarterly*, *28*(1), 6.

Kaufman, L. M. (2009). Data security in the world of cloud computing. *IEEE Security & Privacy*, 7(4).

Kirchmer, M., Franz, P., Lotterer, A., Antonucci, Y., & Laengle, S. (2016). The value-switch for digitalization initiatives: business process management. *BPM-D Whitepaper, Philadelphia, London*.

Kruchten, P., Obbink, H., & Stafford, J. (2006). The past, present, and future for software architecture. *IEEE software*, *23*(2), 22-30.

Lewis, G. A., Cornella-Dorda, S., Place, P., Plakosh, D., Seacord, R. C. (2001). *An enterprise information system data architecture guide* (No. CMU/SEI-2001-TR-018). CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST.

Marr, B. (2006). The Amazing Ways Companies Use Virtual Reality for Business Success. Retrieved November 7, 2018, from https://www.forbes.com/sites/bernardmarr/2017/07/31/the-amazing-ways-companies-use-virtual-reality-for-business-success/#551ad0cf1bae

Mazuryk, T., & Gervautz, M. (1996). Virtual reality-history, applications, technology and future.

Medvidovic, N., & Taylor, R. N. (2010, May). Software architecture: foundations, theory, and practice. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2* (pp. 471-472). ACM.

Milgram, P., Takemura, H., Utsumi, A., & Kishino, F. (1995, December). Augmented reality: A class of displays on the reality-virtuality continuum. In *Telemanipulator and telepresence technologies* (Vol. 2351, pp. 282-293). International Society for Optics and Photonics.

Moore, J. F. (1993). Predators and prey: a new ecology of competition. *Harvard business review*, *71*(3), 75-86.

Pahl, C., Xiong, H., & Walshe, R. (2013, September). A comparison of on-premise to cloud migration approaches. *In European Coference on Service-Oriented Computing* (pp. 212-226). Springer, Berlin, Heidelberg.

Peffers, K., Tuunanen, T., Rothenberger, M. A., Chatterjee, S. (2007). A design science research methodology for information systems research. *Journal of management information systems*, *24*(3), 45-77.

Rebenitsch, L., & Owen, C. (2016). Review on cybersickness in applications and visual displays. *Virtual Reality*, *20*(2), 101-125

Ross, J. W., Weill, P., & Robertson, D. (2006). *Enterprise architecture as strategy: Creating a foundation for business execution*. Harvard Business Press.

Schroeder, R. (2008). Defining virtual worlds and virtual environments. *Journal For Virtual Worlds Research, 1*(1).

Simon, P. (2010). *The next wave of technologies: opportunities in chaos*. John Wiley & Sons.

Slater, M., & Wilbur, S. (1997). A framework for immersive virtual environments (FIVE): Speculations on the role of presence in virtual environments. *Presence: Teleoperators & Virtual Environments, 6(*6), 603-616.

Sowa, J. F., Zachman, J. A. (1992). Extending and formalizing the framework for information systems architecture. *IBM systems journal*, *31*(3), 590-616.

Stark, J. (2016). Product lifecycle management. In *Product Lifecycle Management (Volume 2)* (pp. 1-35). Springer, Cham.

Sudarsan, R., Fenves, S. J., Sriram, R. D., & Wang, F. (2005). A product information modeling framework for product lifecycle management. *Computer-aided design*, *37*(13), 1399-1411.

Taleb, M., & Cherkaoui, O. (2012). Pattern-oriented approach for enterprise architecture: TOGAF framework. *Journal of Software Engineering and Application*, 45-50.

Ternier, S., Klemke, R., Kalz, M., Van Ulzen, P., & Specht, M. (2012). ARLearn: Augmented Reality Meets Augmented Virtuality. *J. UCS*, *18*(15), 2143-2164.

Tupper, C. (2011). *Data architecture: from zen to reality*. Elsevier.

Urbaczewski, L., & Mrdalj, S. (2006). A comparison of enterprise architecture frameworks. *Issues in Information Systems*, *7*(2), 18-23.

Vaishnavi, V., & Kuechler, W. (2004). Design research in information systems.

Vasconcelos, A., Sousa, P., & Tribolet, J. (2007). Information system architecture metrics: an enterprise engineering evaluation approach. *The Electronic Journal Information Systems Evaluation*, *10*(1), 91-122.

Westmark, V. R. (2004, January). A definition for information system survivability. In *System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on* (pp. 10-pp). IEEE.

Wieringa, R. J., Blanken, H. M., Fokkinga, M. M., & Grefen, P. W. (2003, June). Aligning application architecture to the business context. In *International Conference on Advanced Information Systems Engineering* (pp. 209-225). Springer, Berlin, Heidelberg.

Wu, Z. (2009, December). An novel data architecture for data taxonomy. In *Test and Measurement, 2009. ICTM'09. International Conference on* (Vol. 2, pp. 327-330). IEEE.

Yuliana, R., & Rahardjo, B. (2016, April). Designing an agile enterprise architecture for mining company by using TOGAF framework. In *Cyber and IT Service Management, International Conference on* (pp. 1-6). IEEE.

Zachman, J. A. (1987). A framework for information systems architecture. *IBM systems journal*, *26*(3), 276-292.